

Addendum for DeskMate 3.05

Cat. No. 25-1351

- To make the best use of DeskMate's new font capabilities in Draw the following information should be noted:

- If you are using a hard drive system, all fonts must be stored in a single directory. DeskMate's Install program automatically copies all fonts to the same directory, but if you add additional fonts, be sure they are in that directory.
- If you are using a diskette-based system, the diskette containing the fonts must be in the drive so that Draw can access them. (If you are using data diskettes, you can copy your font files (*.ff1) to your data diskettes. You should also copy your printer driver (*dmpdxxxx.res* and *dmpexxxxx.res*) to these diskettes.)
- If Draw cannot find a font, the application substitutes large, block letters on the screen. These are your indication that the requested font was not accessible.
- If you are using MS-DOS version 2.11, you must add the following command to your *autoexec.bat* program so that you can use Draw's font capabilities:

set dmfont=deskmate font path

(If you need information about creating or changing your *autoexec.bat* file, refer to your MS-DOS documentation.)

- For best results, the point size you use for fonts should create characters that are no larger than the size of the screen.
- When you access the Text option from the Options Menu, the dialog box that appears contains choices for Printer or Outline. Use the following criteria to make your choice:

Choose the Printer button to use the fonts built into your printer. This will allow you to print more quickly and with higher quality. You cannot, however, change the point size, width, or color of the characters.

Choose the Outline button if you want the flexibility to make point size, width, or color modifications to your font characters.

- In the Setup option from the F10 Menu, please note the following information about your mouse setting:

- The COM setting you set for your mouse will supersede a COM setting you have previously set for your Communications option. When you access your Communications option, the COM port setting currently used by the mouse will be shadowed so that it is not available for your Communications option.

The page printing function allows mixing of system text and graphics on a page. Currently 10, 12, and 16.7 characters per inch printing is supported. The driver controls the adjusting of graphics on a page to reflect the current cpi - shrinking the graphics to maintain the same aspect ratio as the system text.

General User Functions Resource (DMGUF) :

GUF provides two levels of file I/O as well as the Environment functions and some miscellaneous file handling calls, such as error messages, dialog boxes routines, file-checks, disk queries, etc..

The lowest level supports direct file access such as seeks, reads, and writes but performs no user interaction or error handling other than the processing of critical I/O errors.

The second level supports applications which keep data in a single contiguous block (less than 64K) of memory. This level of file I/O is simple and complete, managing all disk I/O, all possible errors, and full interaction with the user.

The Environment Manager handles the creation, loading, and writing of configuration data files to disk as well as their in-memory management.

DataBase Resource (DMDB) :

The DataBase provides an indexed record manager which contains functions to create files, add, modify, and delete records, fetch single and multiple records, query, sort (stored index & secondary temporary index), and merge data.

The files are made up of application defined "tables" which are defined by their "columns" (fields) and "rows" (records). Each table has its own stored index, columns may contain numeric or character ASCII data which may be labeled as being unique (two records may not contain the same data in the same field).

The database also provides easy access to the Workgroup shared database server for support of multi-user database applications.

Autoload Resource (AUTOLOAD) :

Applications which need a resource to be loaded when Desk is executed use the Autoloader to register the resource's information - resource name, initialization information, and load priority - in the configuration file. Desk uses the Autoloader to load the requested resources. Desk uses the priority information when deciding when the resource should be loaded and unloaded.

Spell Checker Resources (SPELL, SPL) :

The Spell resources is used by the Spell Checker accessory and by application which wish to proof selected text. The spell engine resource, Spl, is the actual spell checker and dictionary which must be licensed separately from the Microlytics company.

Addendum for DeskMate Users

Cat. No. 25-1350

Please note the following changes to your *DeskMate Getting Started* magazine:

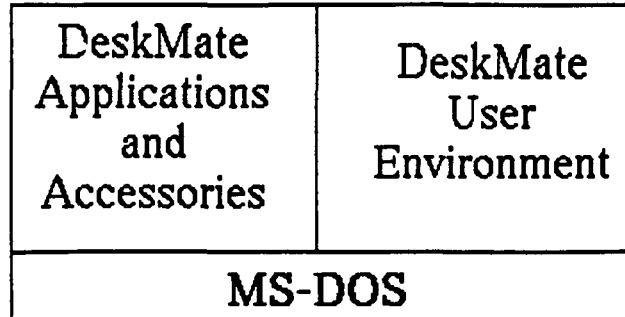
- The "Starting DeskMate" section indicates that the introductory tutorial appears automatically the first time you run DeskMate. This no longer occurs.

However, we highly recommend that you run the introductory tutorial before you begin using DeskMate. To access the tutorial, press TAB until the Teach Me box is highlighted, and then press ENTER. When the list of tutorials appears, the introductory tutorial, DeskMate: An Introduction, is highlighted. Simply press ENTER to begin.

- **For Hard Disk Users** — You need to use the following procedure (instead of the one described in your *Getting Started* magazine) to install DeskMate on your hard disk.
 1. Turn on your monitor and computer.
 2. Insert the DeskMate Diskette 1 into any drive.
 3. If necessary, change to the appropriate drive. For example, if you insert the diskette into Drive A, simply type a : and press ENTER.
 4. Type install and press ENTER.
 5. Follow the instructions as they appear on the screen.

The DeskMate System Architecture

The DeskMate system consists of two major parts - the Desk User Environment and DeskMate Applications and Accessories. The User Environment is made of the System Resources, the Executive (Desk or Runtime), and the DeskMate Bindings.



The DeskMate System Resources (CSR, GUF, DB, etc.) are the individual modules which contain the system functions. These resources are divided by functionality so only those functions required by the application are loaded into memory. See the detail section **DeskMate Functions** which follows for a detailed description of the functionality provided.

The Desk Executive handles the system level tasks which include the loading and unloading of the system resources, applications (both DeskMate and non DeskMate), and accessories. The Executive dynamically links resources to applications, allowing multiple applications to share a resource. Desk allocates and manages the Clipboard. The Executive also performs task-switching, which allows a second application (DeskMate or non-DeskMate) to run while a DeskMate application is loaded. The user may then toggle between both tasks via a "hot-key".

The Bindings are the "bridge" between the DeskMate application and the Desk Executive and the Resources. The DeskMate Library (*dm.lib* and *dmmed.lib*) contains the entry points to all functions in the Executive and the Resources. When an application wishes to use one of these functions it informs the Executive which in turn loads the function into memory and dynamically links it to the application.

DeskMate applications are the controlling modules in the system. DeskMate applications are "event driven", accepting an input command from the user and acting upon it. The Core Services Resource provides advanced user interface features which allow an applications to register its graphic components (listboxes, pushbuttons, etc.), then it manages these components returning the user actions to the application as single events. Event categories include Character (keyboard), Mouse, Command (menubar functions and components), Application (task-switch & accessories), as well as special purpose events required by the system and available to the application.

Memory Map Example

DeskMate runs in a range of system memory configurations from a minimum of 384K on a RAM machine to the maximum of 640K on a ROM machine. The following picture illustrates the memory map for a 640K RAM system configuration.

Tandy 4000 using MS-DOS 3.2, 640K of memory, and EGA video :

<u>Arena Owner</u>	<u>Size in Bytes</u>	
DMDB.RES	60768	
DMCSR.RES	4896	*Screen Save Buffer
DMMDSERI.RES	1824	
DMVDEGA.RES	22368	*Video Driver
DMGUF.RES	30416	
DMCSR.RES	76896	
Clipboard	16400	(8K in 384K configuration)
DESK.EXE	18528	

<u>*Screen Save Buffer Sizes</u>		<u>*Video Driver Sizes</u>	
CGA	3048	DMVDCGA	21K
1000	6048	DMVD1000	22K
EGA	4896	DMVDEGA	22K
HERC	6064	DMVDHERC	24K
TC16	12096	DMVDTC16	20K
VGA	0	DMVDVGA	21K

DeskMate Functions

Executive (DESK or RUNTIME) :

The Executive handles the DeskMate system level tasks such as loading and unloading of resources, applications, and accessories.

The Clipboard allows the user to transfer data from one DeskMate application to another, or solely within an application. The DeskMate Clipboard can contain any type of ASCII, binary, or mixed data which the application requires. The Clipboard remains intact at all times while DeskMate is active and is available at the application's request.

Application chaining is supported through the capability of setting the next application to execute after the current application terminates.

Core Services Resource (DMCSR) :

The CSR handles the user interface portion of an application including video output (graphics, system and stroke font text, and components), keyboard and mouse input, printing, and communications.

All video graphics are drawn onto an 8000 x 5500 world coordinate plane. World coordinates are automatically mapped to the resolution of the video device by the currently loaded video driver so that one version of an application can be written to run on any video resolution. Normalized world coordinates allow the application to perform exact pixel-by-pixel graphics.

Windows, viewports, and clipping regions are supported by the CSR. Screen saving/restoring, scrolling, filling, and clearing functions are provided.

Character (system and stroke font) and graphic outputs are supported. Bold and underlined character attributes are currently supported. Graphics functions include point, line, rectangle, beveled rectangle, ellipse, polygon, polyline, arc, and bitmaps. Color, line style, and pattern attributes are supported. The "Forms Manager" provides the functionality required to store, display, and manipulate (move, size, find, change, reorder, etc.) graphics elements on the screen as well as transfer the information via the clipboard to other applications.

Currently the Tandy 1000, CGA, EGA, VGA, MCGA, and Hercules video modes are supported. The video mode is automatically detected and the appropriate driver loaded or the user may override the automatic detection by selecting the driver to load (dmvid.exe).

The Component, Window, and Event Managers process the applications components, funneling the user actions back to the application as events.

The Dialog Box Manager controls execution of dialog boxes, returning to the application as necessary for information updating. Cursoring through the components, redrawing components, and handling pushbuttons are all done by the resource.

Message Boxes display word-wrapped messages, saving and restoring the screen under the box for the application.

The printer functions allow both page printing, handling printing to the screen, printer, or file for the application and direct printing which gives the application full control. Currently ASCII, IBM graphics, Tandy, Daisy Wheel, and LaserJet printer drivers are provided.

General Design of a DeskMate Application

DeskMate applications are typically "transaction centers", user inputs are processed and results are displayed. The application usually contains its initialization phase, its "event-loop" where user input is accepted, and its clean-up phase when the user chooses to exit the application.

The application is always in control, it decides when it is ready to receive input and call the event read function. For example, when the user selects a command from the application menubar, the system handles the user interface and returns the selected item to the application for processing.

The following is a general outline of a DeskMate application's main processing loop.

```
main()
{
    /* Initialization Phase */
    bind to any resources;
    create any child windows in the work area;
    draw the menubar and any default information;
    open any file passed on the parameter line;

    do
    {
        read the event;

        switch( type of event )
        {
            case is command event :
                /* the user selected a menu option, */
                /* pushed a button, etc. */
                ProcessCommand( event parameter );
                break;

            case is character event :
                /* the user entered an alpha-numeric */
                ProcessKeystroke( event parameter );
                break;

            case is mouse event :
                /* the user positioned the mouse, */
                /* started a selection, double-clicked */
                ProcessMouse( event );
                break;

            case is an application event :
                /* the user selected an accessory or task switching */
                run the accessory or task switch;
                break;

        } /* end of switch on type of event */

    } while ( user has not chosen quit );

    /* Clean Up Phase */
    release any resources;
    exit;
} /* end of application main module */
```




Tandy Electronics

A DIVISION OF TANDY CORPORATION

RESEARCH AND DEVELOPMENT

1300 Two Tandy Center
Fort Worth, TX 76102
Telephone (817) 390-2181
Fax (817) 878-6575

November 20, 1989

Dear DeskMate Developer,

We are building the new DeskMate Development System, the kit, and would like to give you an opportunity to review some of the new information presented in the kit that may affect your development efforts. The enclosed documentation appears in the new About This Kit and DeskMate Development Guide manuals. Please review this documentation and make any comments as soon as possible.

You may mail your responses to:

Attn: DeskMate Support Services
Tandy Electronics
1300 Two Tandy Center
Fort Worth, Texas 76102

or FAX your responses to: (817) 390-2964.

Sincerely,

Rachel McKenzie
Manager, DeskMate Support Services

cc: S. Cutler
H. Elias
D. Tanner
G. Schenberg

System Overview

DeskMate 3 was introduced in the fall of 1988. This version of DeskMate enabled developers to write applications for the interface and environment. This software is referred to as DeskMate 3.0 (or simply 3.0) and was not compatible with previous versions of the DeskMate product. DeskMate 3.0 includes the following versions:

1000 SL	DeskMate 03.00.00
1000 TL	DeskMate 03.00.00
Retail	DeskMate 03.00.00 and DeskMate 03.02.00
Runtime	DeskMate 03.02.01

This year's DeskMate 3 product is version 3.3 which is DeskMate 3.0 compatible. DeskMate 3.3 includes the following versions:

1000 SL/2	DeskMate 03.03.00
1000 TL/2	DeskMate 03.03.00
Retail	DeskMate 03.03.01
Runtime	DeskMate 03.03.01

Technical Overview

The DeskMate 3 environment consists of the *executive* and the *resources* that contain the system functions. These resources provide the user interface, three levels of file input and output including a database, and printing support for the application.

DESK.EXE, the executive, loads and unloads DeskMate programs - *applications*, *accessories*, and *resources*, as well as non-DeskMate applications. The executive expects DeskMate programs to contain specific information in the program header used when the program is loaded. Refer to the DESKHDR.EXE documentation in the DeskMate Development Guide, Tools and Utilities section for more information.

DeskMate applications (.PDM extension) are the controlling modules in the environment.

Accessories (.ACC extension) are mini-applications which can pop-up over an application. Accessories are small programs that perform very specific tasks for the user and have functionality in several applications.

DeskMate resources (.RES extension) are the work-horses of the environment. They provide the common functionality required by most applications - user interface, file i/o, communications, printing, etc.. These resources are *terminate and stay-resident* (TSR) programs which have a focused scope of functionality. They may be shared by more than one program at a time. Resources are also used to provide device-specific functionality determined at execution time, such as, video and printer drivers.

All DeskMate programs must link with a DeskMate library to make use of functionality in the executive or any of the resources. The DeskMate libraries, DM.LIB and DMMED.LIB, contain the *bindings*, or bridge, used by a program to call a function in the resource.

When a program requires functionality provided by a resource, it *binds* to the resource by calling the executive and requesting the resource. The program binds to the resource once, before calling any functions in the resource. The bind call requests the executive to 1) find and load the

resource and 2) inform the application of where the resource was loaded. When the executive loads the resource it resolves the resource's *service request vector* (srqv) in the bindings - the far address of the resource's entry point, and increments the resource's *use count*. The vector is used to make far calls from the application into the resource. The use count allows the executive to determine how many programs are using the same resource. When the program no longer requires the resource, it *freed* the resource. The executive decrements the resources' use count. The resources is not unloaded until all programs using the resource have freed the resource (its use count is zero) and the memory is needed to load another program.

The DeskMate 3.0 system used different file extensions to distinguish between the product resources (RES) and the runtime resources (RRS). Which resource was loaded is determined by the executive, DESK.EXE or customized RUNTIME.EXE, respectively.

The DeskMate 3.3 system uses the same file extension (RES) for both the product and the runtime resources. DeskMate system resources which must be used by the 3.3 system use the R89 extension to differentiate them from the 3.0 resources. The executive determines which resource to load.

These file conventions only affect developers who write their own resources. Resources developed for the 3.0 system have to be shipped with RES and RRS extensions for installation on a product or DeskMate 3.2 runtime. Resources developed for the DeskMate 3.3 system use the RES extension.

The executive, resources, and Setup, Page Setup, and Help accessories are distributed in the DeskMate runtime. These files are referred to as the *DeskMate system files*. Chapter 5 of this manual details how to obtain the runtime software.

The DeskMate executive and resources include:

DESK.EXE	The Executive handles the system level tasks for the environment and the applications.
	DESK Managers include: Clipboard Manager Desk Executive
DBBUILD.RES	This Database Resource allows applications to create database files.
DBREAD.RES	This Database Resource allows applications to open and read database files.
DBUPDATE.RES	This Database Resource allows applications to open, read, and update database files.

DMCSR.R89 The Core Services Resource handles the user interface and "core" functionality for applications. This resource is always used by a DeskMate application.

CSR Managers include:

- Communication Manager
- Component Manager
- Configuration Manager
- Dialog Box Manager
- Event Manager
- Information Box Manager
- Keyboard Manager
- Menu bar Manager
- Message Box Manager
- Mouse Manager
- Print Managers
- Titleline Manager
- Video Manager
- Window Manager

DMDB.R89 This Database Resource provides database file access for applications developed with the 3.2 development system on a 3.3 system.

DMGUF.R89 The General User Functions Resource provides high- and low-level file I/O functionality. See the Executive section for details on what functionality each GUF resource provides.

GUF Managers include:

- Environment Manager
- File I/O Manager

DMFORM.RES The Form Manager Resource provides vector graphics and stroke font support for applications.

DMMD*.RES The pointing device drivers JOY (Tandy 1000 joystick), SERI (serial mouse), P (Micro-Channel mouse)

DMPD*.RES The printer drivers ASCI (ASCII), IBMM (IBM-compatible graphics), LASR (Laserjet-compatible), 1 (DMP 105), 2 (DMP 200, 420, 430)

DMVS*.RES The 80 column video drivers CGA, 1000 (TGA), TC16 (ETGA), EGA, VGA, MCGA, and HERC. The 40 column video drivers LRES, TC40, T256, H (Hercules), E (EGA), and M (Monochrome EGA)

PRGUF.RES The Power and Run General User Functions Resource, provides a subset of the GUF functionality. See the Executive section for details on what functionality each GUF resource provides.

The DeskMate libraries contain functionality which is linked into the application. The Library Manager lists all functions available in the libraries.

DeskMate 3.3 Modifications and Enhancements

The DeskMate 3.3 system added the following modifications or enhancements to the user interface and environment. For more information about how these changes affect your application's compatibility on all DeskMate 3 systems, refer to the DeskMate Development Guide, Getting Started section. For more information about the user interface changes refer to the DeskMate Style Guide. For more information about the enhanced calls, see the appropriate section in the DeskMate Technical Reference; the new calls are clearly marked as "1989 DeskMate 03.03.0x ONLY". For information on the new utilities mentioned, refer to the DeskMate Development Guide, Tools and Utilities section.

User Interface

The title-line was rearranged to have the Help F1 prompt appear at the far left over the function key on the keyboard. The time indicator was moved to the far right.

In dialog boxes and message boxes, the default push button - button which will be pressed by the Enter key - appears with a dashed box around it to notify the user.

The "Sticky Menu" interface was added to the user interface. See the Development Guide, Getting Started section for more information if your application was predicting menu bar events.

Grayed menu items can be highlighted by the user, although they are not selectable (enhancement added for new help).

The busy icon is now animated, it cycles through a pattern.

An *About* menu option standard was adopted for all applications.

The "arrow algorithm" used in dialog boxes was optimized to eliminate "dead spots", components in a dialog box that could not be accessed via the arrow keys.

Environment

Applications written with the DeskMate 3.0 system used SETHEAP.EXE to set their minimum and maximum load size requirements. DeskMate 3.3 applications should use the new DESKHDR.EXE utility which also contains the program's split allocation, code shed size, and version number. The 3.3 executive will use the information stored in the new header when loading a program and when deciding how much code to shed when more space is needed to load an accessory. The 3.0 executive will ignore the extra information.

Additional accessories can now be accessed through the *More* option on the F10 Menu.

Context-sensitive help is now available in all pop-ups, dialog boxes, message boxes, and accessories, and on menu items.

The Form Manager which was part of the Core Services Resource, DMCSR.RES, was split out as a resource, DMFORM.RES. The CSR on a DeskMate 3.3 system will automatically load the resource on a **form_open** call for a 3.0 application.

The General User Functions Resource was split into two resources, DMGUF.R89 and PRGUF.RES.

The Core Services Resource, CSR, only saves the first six (6) colors in the configuration file. Colors 7 through 16 are now considered to be application specific. The application must decide whether to save the colors with the data file or in its own configuration file.

The **dm_file_search** function honors diskette label files created with DMLABEL.PDM, a DeskMate utility.

The **dm_file_search** function no longer prompts the user to search the entire system before doing so.

The printer drivers were enhanced to support new line style widths, the patterns were changed to match the video drivers, the maximum number of characters printed on a line was increased, and several printing problems were fixed. For a complete list of changes, see the DeskMate Development Guide, Getting Started section.

A set of new video drivers was added, allowing applications to "video swap" into a 40 column screen resolution. These drivers require a DeskMate 3.3 system.

The Runtime executive now supports parameter passing to the runtime application and the execution of the runtime module from a DeskMate 3.0 DeskTop. This feature is important to applications which require the DeskMate 3.3 system to operate.

Application Data Files

In data files for the DeskMate 3.3 product have the same format as was used in the previous DeskMate 3 versions. The Address Book data file lengthened the Title and Address fields. Refer to Appendix A, DeskMate 3 File Formats in the DeskMate Development Guide for details.

Introduction

After reading About This Kit, reviewing the DeskMate Style Guide, and installing your DeskMate 3 product and development system you are ready to develop a DeskMate application. Before you begin development, we should review the key information discussed so far and introduce some new topics which you should find beneficial in the development of your application.

The Kit contains the 1) development files, 2) samples, and 3) tools and utilities need to develop a DeskMate applications. The DeskMate Technical Reference defines every function call available in the DeskMate libraries.

DeskMate applications are primarily written in C but may also be written in assembly language. Programs may be written in any of the memory models but only the small and medium memory models have DeskMate libraries. Refer to Memory Models and Development Tools, in this section, for a detailed discussion of memory models, and compiling, linking, and debugging of DeskMate applications.

The DeskMate Style Guide defines the DeskMate User Interface. DeskMate applications use menus, dialog boxes, message boxes, and interface components to communicate with the user. DeskMate applications support both a keyboard and mouse interface. Your application should meet the DeskMate standards defined in this guide.

From the System Overview in About This Kit, you learned about DESK, the DeskMate Executive, and the key DeskMate resource - Core Services Resource (Core or CSR), and the other resources available in the DeskMate environment. Applications communicate with these resources through the DeskMate libraries.

There are now two versions of DeskMate 3 in distribution, DeskMate 3.0 (includes 3.2) and DeskMate 3.3. Your application should check the system version number, when it is initially loaded, by calling `dm_inquire_product` to determine which version of the environment the application is running on.

Now, let's introduce some new DeskMate programming topics.

DeskMate uses a *world coordinate system* to access the video. In the programming examples and the function call descriptions in the DeskMate Technical Reference you will often see the defines, `CHAR_XEXT` and `CHAR_YEXT` used. These defines allow the programmer to reference points on the screen as character locations. DeskMate also allows the video to be accessed at a pixel or device level. See DeskMate Coordinate Systems for a detailed discussion about world and device coordinates.

DeskMate applications are *event-driven*, they wait for the user to perform an action and then act upon the action. The CSR provides an Event Interpreter or Manager which translates the user's actions into events the application can process. Applications can write their own event interpreters to capture events before and after the CSR's Event Manager has handled them. For more information, see the Event Manager section of the DeskMate Technical Reference.

DeskMate allows mini-applications, called accessories, to pop-up over the current application. When there is not enough available memory to load the accessory, Desk will try to make room for the accessory by getting rid of part of the application's code and moving the rest. This process is referred to as *code shedding*. The following criteria is used to determine if your application can be code shed to run an accessory. If your application cannot be code shed then it MUST call `dm_exec_dont_shed` when initially loaded to insure that it is not code shed to run an accessory. Your application should also set the code shed size using the DeskMate utility DESKHDR.EXE.

- 1) An overlaid application cannot be code shed since it cannot be guaranteed that it will be restored from the disk in the same configuration it was in before the accessory was run.
- 2) An Application which uses event interpreters or interrupt handlers cannot be code shed because the interpreters and handlers are address dependent. When the application is moved during the code shed, the handlers are moved and may no longer function correctly.

Note: On a DeskMate 3.3 system the application may be able to code shed if the handlers are placed in the IMPURE segment which is not altered during a code shed. Refer to the detailed information for DESKHDR.EXE in the Tools and Utilities section of this guide for more information about splitting applications.

- 3) A medium or large model application which has too many fix-ups (more than 200), cannot code shed in a DeskMate 3.0 system but can on a 3.3 systems which supports unlimited fix-ups.

Note: This deficiency in the 3.0 system can be overcome by naming the code segments and limiting the number of code segments used to a smaller number. Refer to your compiler documentation for more information about overriding the default code segment name.

The executive and the resources often use the application's stack. The CSR and its drivers require the application stack for busy icon and mouse processing. A *packed executable* has a very small temporary stack while it is being loaded before the stack is expanded. This stack can be overflowed during the loading of the application if the busy icon or mouse processing consume more of the stack than is available. You should not pack your DeskMate executable and should allow at least 2048 bytes of stack space for the executive and DeskMate resources, and 4096 bytes if the Form Manager Resource is used.

Compatibility and Programming Issues

Runtime Executive

The 3.3 runtime executive allows an application to be launched from a 3.0 DeskTop as a runtime. This enhancement was added for applications which require the 3.3 environment to operate in but still want to be launched from 3.0 DeskMate products. An application making use of the new 40 column video drivers would be an example of an application requiring the 3.3 system.

To run from a 3.0 DeskTop your application can provide a small "compatibility" application which checks the current system and then runs the application. The compatibility check should also be performed within your application in case the user executes the application from the DeskTop. If your application is large, you should consider providing the compatibility application since it will take less time to load and unload it rather than your application.

The function **dm_compat**, a DeskMate Library function, checks the version of desk currently running and decides if the application

can run on the system.
cannot run because the user is task-switched.
needs to run from the new runtime.

The compatibility application calls **dm_compat**, sending it the name of your customized runtime module, and checks the return code and handles it as follows:

```
main()
{
    int      product_info;
    char     RuntimeName[] = "VENDOR.EXE";

    product_info = dm_compat(&RuntimeName[0]);
    if ((product_info & DM_VERSION) == 0)
    {
        if ((product_info & DM_COMPAT_FLAG) == 0)
        {
            csr_init();
            display "Cannot run while task-switched."
            csr_end();
        }
    } /* running on a 3.0 system */
    else
        /* running on a 3.3 system */
        dm_SetNextApp( to VENDOR.PDM );

    exit();
} /* end of compatibility application */
```

If the application is running on a DeskMate 3.0 system and is not in a task-switched context, then **dm_compat** will call **dm_SetNextApp** to your application's 3.3 runtime. The compatibility application will either cause the application to run on the current system or as a runtime or inform the user that the application cannot run in the current context.

The Help System

DeskMate 3.0 Operation

Help is provided through an accessory. Application help is therefore only available when an accessory can be executed. The application always knows when the user requests help. Applications can write their own event interpreters to capture the F1 key and provide the user with the level of context-sensitive help they deem appropriate.

DeskMate 3.3 Operation

Context-sensitive help is now provided through an Intelligent Help Manager which captures the context of the application and gives specific and general help, specific to the application state. Help is now available in pop-ups, including accessories, and while the menu bar is being accessed. Help may be given at any time, for instance while the user is in a dialog box, and the application is not always aware of when the user requests help. The application can register call-back functions which will be called prior to and after help is given. Refer to the Help Manager section of the Technical Reference for more information.

Compatibility Issues

For applications written for the DeskMate 3.0 system, running on a 3.3 system:

In applications which are not providing any context-sensitive help (by trapping the F1 key), or are not providing help for all the new context possibilities, the user will get a message stating that help is not available. The developer can decide if this is acceptable or do one of following to ensure the user is always presented with help in any DeskMate 3 system.

1) Distribute a Help Compatible System consisting of:

- a) An application help data file.
- b) The help compatibility accessory, DMHELP88.ACC.
- c) The DeskMate 3.3 Intelligent Help Manager, DMHELP.ACC and DMHELPENG.RES.

Upgrade DeskMate 3.0 user's DMHELP.ACC file with the new Intelligent Help Manager, see the Distributing Your Application section in this manual for more information. The new help accessory will chain to the compatibility accessory and provide general application help from the help data file on the upgraded 3.0 system and context-sensitive help on a DeskMate 3.3 system.

- 2) Handle the new areas of context-sensitive help by using an event interpreter and trapping the F1 key. Refer to the Event Manager section of the Technical Reference for details about writing an event interpreter.

The F10 Tandy Menu

The user can now run new accessories from the More option on the F10 menu or from an upgraded Setup accessory. To run accessories on all DeskMate 3 systems, your application should not perform any range checking on the accessory value before running the accessory. The F10 menu distribution, number of items and their names, varies from

system to system depending on the capabilities of the DeskMate system. Your application should not make any exceptions or assumptions when running accessories, it should simply run the accessory the user requested.

Code Shedding when Running Accessories

DeskMate 3.0 Code Shed Operation

In this environment when an accessory does not fit, the executive code sheds 32K of the application. Applications which can not have their code shed and replaced from disk called `dm_exec_dont_shed`. See the discussion of code shedding in the Introduction of this section for a discussion of code shed criteria.

DeskMate 3.3 Code Shed Operation

In this environment the amount of code shed space for an application is stored in the application's header built by `DESKHDR.EXE`, the DeskMate utility. The executive looks at this information to determine how much, if any, of the application to shed in order to load the accessory. If the code shed size is less than 32K, applications should call `dm_exec_dont_shed` to register that information with the DeskMate 3.0 executive.

Programming and Compatibility Issues

Your application may not function properly if the application cannot be code shed and it does not inform the executive by either setting the code shed size using `DESKHDR.EXE` and/or by calling `dm_exec_dont_shed`.

Your application will not function properly if does its own code shedding to make room for an accessory for the following reasons.

- 1) The DeskMate 3.0 accessories were generally less than 32K, so most accessories would run if that amount of memory was available. In the 3.3 system, most of the accessories use more than 32K. Freeing a specific amount of memory will probably not cover all cases.
- 2) Accessories can load one or more resources when they run. Depending on the function of the accessory, the resource may stay loaded after the accessory exits. For instance, the Spell Checker allows the user to turn on auto-proofing and exit the accessory. The spell resource stays resident to handle the auto-proof function. Your application will not be able to recover the memory it freed for the accessory.
- 3) New accessories may be executed through the new More option, your application cannot predict how these new accessories will operate or how much memory they will require.

If there isn't enough room to load an accessory, the executive will warn the user. It is better not to run an accessory, than to run an accessory and not recover properly.

To run accessories on all DeskMate 3 systems, your application should do the following:

- 1) Set the code shed size (0 up to code size) for your application using DESKHDR.EXE.
- 2) If the code shed size is less than 32K, call **dm_exec_dont_shed** on a DeskMate 3.0 system.
- 3) For applications which use all available memory and cannot be code shed, consider doing one or more of the following:
 - a) shed data which can be regenerated after returning from the accessory.
 - b) shrink the unused data size to free memory for the accessory. Your application must handle not being able to expand out the data if the memory is no longer available.
 - c) free resources which can be reloaded after returning from the accessory. Your application must handle not being able to reload the resources if the memory is no longer available.

"Sticky Menus" and Selectable Grayed Menu Items

Since the menu bar processing is done within the DeskMate environment, this enhancement is transparent to the application. Applications which use their own event interpreters and are predicting the state of the menu bar based on the mouse or arrow events are affected by this change.

In the DeskMate 3.0 system, a single mouse click did not affect the state of a menu bar. In the 3.3 system, a single mouse click can cause a menu to drop or will change the selection of a menu item.

In the 3.0 system, the up and down arrows skipped over grayed menu items. In the 3.3 system, the up and down arrows do not skip grayed menu items.

To be compatible on all DeskMate 3 systems, applications which predict user events must handle the differences in the menu bar user interface in each system. To aid the developer, the new **mb_get_status** call was added to get menu bar status information.

Animated Busy Icon

The Tandy busy icon is now animated. The icon processing can cause problems for applications which are accessing video memory directly and are making timing assumptions about the busy icon. If your application meets this criteria, make sure your application disables the busy icon while it is accessing video memory.

Form Manager and GUF Resource

Loading of the Resources for 3.0 Applications

The **DMFORM.RES** is automatically loaded on the first **form_open** call. Both GUF resources, **DMGUF.R89** and **PRGUF.RES** are loaded with the **guf_bind_init** call.

If the resource does not fit in available memory or the resource file could not be found, the **form_open** and **guf_bind_init** calls will return an error. You should ensure your application is checking the return code from both call and handles the conditions properly.

If your application uses all available memory, the **form_open** call should be made **BEFORE** all of memory is allocated.

Loading of the Resources for 3.3 Applications

The new binding call for the Form Manager resource, **csr_form_bind_init** will return an informative error **DM_EXISTS** if the application is running on a 3.0 system.

Both GUF resources, **DMGUF.R89** and **PRGUF.RES** are loaded with the **guf_bind_init** call. To load only the **PRGUF.RES** resource, call **prguf_bind_init**.

Video Drivers

Driver Names

The DeskMate 3.0 video drivers used the **DMVD** prefix, the 3.3 drivers use the **DMVS** prefix. The video drivers must match the version of the CSR being used, mixing of systems is not allowed. Applications using the **cfg_get_vid_driver** call to determine what video driver is loaded are affected by this change and should handle the differences in the systems.

Video Detection

The **VGA** video driver, **DMVDVGA.RES**, incorrectly returned **VID_EGA** in the **VID_DEVICE.card** element when the **vid_inquire_device** call was made. In order to determine if the video was in fact **VGA**, the calling program compared the **VID_DEVICE.dc_yext** element to 480. The **VGA** video driver, **DMVSVGA.RES**, correctly returns **VID_VGA** from the **vid_inquire_device** call. If your application makes use of the **vid_inquire_device** call, you should ensure you handle the differences appropriately.

Palettes

The **DMVSVGA** driver uses different palettes than those used by the **DMVDVGA** driver. If your application accesses the palette information directly, then your application will exhibit different default color settings in the 3.0 and 3.3 environments.

Printer Drivers

Line Styles

The line widths, LINE_WIDTH1 and LINE_WIDTH2 are now supported for the dotted, dashed, and dot-dash line styles. These widths were only supported for LINE_WIDTH1 which exhibited printing problems when a line crossed a print band.

The line style DENSE_DASHED is now supported by the printer drivers.

The thickness of the wider line widths was changed to match the world coordinate width used by the video drivers.

LINE_WIDTH1	1 pixel wide
LINE_WIDTH2	"best look", normally 2 pixels wide
LINE_WIDTH3	50 world coordinates wide
LINE_WIDTH4	75 world coordinates wide
LINE_WIDTH5	100 world coordinates wide

Print regions

The 132 character maximum line has been removed and now as many characters as will fit into the width of the print band will be printed. The width of the print band for printers with a wide carriage is 13200 world coordinates. This translates to the following number of characters depending on the current character per inch setting:

10 CPI	132 characters
12 CPI	158 characters
condensed	220 characters

The dimensions of the printable region for the 3.0 printer drivers was sometimes less than 8 x 11 1/2 inches. The 3.3 printer drivers now print exactly to 8 x 11 1/2 inches. This apply to IBM-compatible graphics printers. The Tandy 2100P with micro line-feed control prints a page 11 3/8 inches instead of 11 1/2. Other non-Tandy printers exhibit the same incompatibility.

The quarter-inch on the left and right side of the paper is the default "unprintable region" for printers. The laser printer has its own specific unprintable region.

Landscape printing

The DeskMate 3.0 drivers did not do a form feed at the end of a landscape printed page, the new drivers do.

The DeskMate Checklist

Programs that will be sold by Radio Shack as DeskMate applications must meet these requirements:

1. The program must be implemented using the DeskMate Development System and use the DeskMate environment.
2. The program must be installable using the DeskTop's F7 Menu, Install option. Refer to the Installation and Upgrade Procedures section which follows this Checklist for more information.
3. The program must support the DeskMate 3.2 and use the DeskMate 3.3 help system.
4. The program should run all accessories (including "More...") and have the F10 menu button on its menu bar.
5. The program must permit task switching from the F10 menu.
6. If the program uses a cut/copy/paste function, the program should support the DeskMate clipboard as its cut/copy/paste buffer. If the program has graphics capabilities, it should use the DeskMate Forms Manager to permit the data to be transferred in the DeskMate graphics format.
7. The program must have the F9 notification menu button enabled.
8. If the program changes the user-defined colors, the program must restore the colors to those specified by the user when the program terminates.
9. The program must not use DOS overlays. If new portions of code must be overlaid onto an executing program, the program should use DeskMate Resources instead of overlays.
- 10. The program should use the DeskMate printer drivers. If a specific driver is not available, the printer development kit should be used to write the necessary driver.
11. The product must be submitted to Radio Shack Computer Merchandising for interface and style guide approval before the application can bear the trademarked DeskMate User Interface logo.
12. The product package should display the trademarked DeskMate User Interface logo.

Installation and Upgrade Procedures

All DeskMate applications should have an installation program which is itself a DeskMate application. The installation program should be easy to use and not alter the user's system without the user being notified.

The installation program should not perform DOS commands which might alter the user system (other than creating directories and copying files), such as setting the date and time, deleting AUTOEXEC.BAT or CONFIG.SYS files, modifying AUTOEXEC.BAT or CONFIG.SYS files such that the user cannot easily recover.

Whenever appropriate, the user should be given a choice to continue the process or cancel. For instance, if the installation is about to delete all system files from a previous version of your product, the program should inform the user giving the option to approve or cancel the process.

Every application must:

Have an INSTALL.EXE program which launches the application's INSTALL.PDM file from a DeskMate 3.3 runtime. This program is used to install stand-alone versions of a product. Use the INSTBLD.PDM program to build your customized INSTALL.EXE program. This file must be on the same diskette as the application's customized version of RUNTIME.EXE.

Have an INSTALL.PDM application which copies files to the user's hard disk using the following guidelines:

Create a directory for the user in which the files are installed. Present the user with a default pathname which can be modified.

When installing on a DeskMate product, do not copy the DeskMate system files unless an upgrade has been recommended or your product requires the version of your runtime. The installation program should determine the DeskMate version by the method outlined in the Determining the DeskMate Product Version section which follows.

To install as stand-alone system copy the DeskMate system files along with your application's files to the directory.

If your product uses the DeskMate Help system, copy the application help file to the directory along with the application.

Neither the INSTALL.PDM nor INSTALL.EXE files should be copied to the hard disk.

For 40 column applications, the INSTALL.PDM must also be a 40 column application.

Do not install the DeskMate system files in a directory which contains a DeskMate product or another vendor's runtime, unless Tandy has recommended you upgrade a system due to an incompatibility or to fix a known problem. A runtime installation should never downgrade/upgrade a user's DeskMate system as it might inadvertently cause the system to no longer function properly. Check

the version number of a file before upgrading the user to ensure the user's product is not mistakenly downgraded.

Provide a DESKTOPD.CFG configuration file which is used by the Desktop install function. This file should be copied along with the other application files during the installation.

Create this file using the DeskTop Menu (F7) Createauto option.

DESKTOPD.CFG is used by the AUTOCONFIG application box on the DeskTop. When the user changes directories to your directory, the box will change to show your application and list of data files.

Have a diskette label file, LABEL.LBL, created with DMLABEL.PDM which contains diskette information used in file searching and for diskette prompts. The file also contains instruction flags for each file which tell the installation program how to copy the file. The diskettes must also have unique volume id's used by your customized INSTALL.EXE and the file search function to prompt for diskettes. The diskette label file should not be copied to the hard disk.

To provide help during the installation process, an INSTALL.HLP help file can be supplied with the product. This file must reside with INSTALL.PDM and should not be copied to the hard disk. If you choose not to provide help during the installation process then this file is not needed.

The user documentation for installation on a DeskMate DeskTop should give the following directions:

- 1) The user should be told to insert the diskette (use the name from the label program) which contains INSTALL.PDM into any floppy drive.
- 2) Direct the user to use the Desktop Menu (F7) Install option to install your application on the DeskTop.
- 3) The user should then follow the prompts given by your installation program.

To reinstall or upgrade on a DeskMate 3.2 system, the user should be instructed to use the Desktop Menu (F7) Delete option to remove the application's definition and then follow the installation directions outlined above.

The user documentation for installation or upgrade of a stand-alone system should direct the user to:

- 1) The user should be told to insert the diskette (use the name from the label program) which contains INSTALL.EXE into any floppy drive.
- 2) Direct the user to change to that drive.
- 3) The user should then run INSTALL.EXE to do the installation.

If your application allows the user to make backups of the product diskettes, then the user should be directed to use the DISKCOPY command to insure the volume id's are copied when the diskettes are copied.

Determining DeskMate Product Versions

Installation launched from the DeskTop

The installation program, INSTALL.PDM, can detect if it was invoked from the DeskTop through the F7 Install option by calling **env_open** with the following ENVDATA structure.

```
ENVDATA your_env =  
{  
    "USER.CFG",  
    "DMCONFIG",  
    "ENV NO CREATE",  
    "USER",  
    (char far *)0,  
    0,  
};
```

If **env_open** does not return **DM_ERROR**, then install was invoked from the DeskTop.

If invoked from the Desktop, do the following to determine the DeskMate version:

```
ret code = dm_inquire_product();  
if (ret code & DM_VERSION) != 0)  
    user has DeskMate 3.3  
else  
    user has DeskMate 3.2 or less
```

Before copying the necessary files (based on the DeskMate version) to a directory you must make sure the DeskMate product is not in that directory. If none of these files are found, then DeskMate is not in the directory.

- 1) Ensure **DESK.EXE** is not present.
- 2) If it is not present, then check for a Tandy ROM machine in which the file is in ROM. Check that at least three of the following files are not in the directory, since it is possible that one of these applications may be in ROM:

```
ADDRESS.PDM  
CALENDAR.PDM  
FORMSET.PDM  
FILER.PDM  
DRAW.PDM  
TEXT.PDM
```

If your runtime executive and application files are present, you can consider this installation to be an upgrade and copy the files.

If your executive and application files are not present, see if any DeskMate 3.0 runtime resource, .RRS extension, or your runtime files are present. If so, there is another runtime application in that directory and you should not install in this directory.

Installation launched from the INSTALL.EXE

If INSTALL.EXE invoked INSTALL.PDM, you have to search the system to determine if DeskMate is present.

```
ret_code = dm_file_search("DESK.EXE", pPathbuffer, 0);
if (ret_code == 1)
    The file was found and the path is in pPathbuffer
else
    The file was not found, so call dm_file_search for the
    DeskMate application files listed above.
```

If none of these files are found, the user does not have the DeskMate product.

How to get the file version

As long as your application, accessory, and resource files use the DESKHDR.EXE utility, you can determine the version of your files in the manner described below. Do the following to determine the version of the file:

- 1) Open the file, refer to this FileHeader structure for variable offsets.

```
struct FileHeader
{
    int      MagicBytes[12];
    int      RelocSeek;
    int      VersionNum;
    char     DM89Key [4];
};
```

- 2) The element RelocSeek must be greater than 25H.
- 3) The element DM89Key must contain the four bytes "DM89".

If items 2 and 3 are met, then the DeskMate file is version 3.3 or greater. This method can be used by your INSTALL.PDM for upgrading only files with prior versions. The element VersionNum contains the DeskMate 3.3 (or greater) version number. The format of this element is file dependent, for DeskMate resource files, *.RES, the version number is binary. DeskMate application and accessory files use ASCII version numbers.

Runtime Distribution Guidelines

Only distribute files listed in Exhibit A of the DeskMate Distribution License. Files marked for non-distribution should not be distributed.

Do not distribute mixed versions of the DeskMate system files. For instance, do not distribute the 3.2 versions of any of the accessories with the 3.3 resources or vice versa.

Files which MUST be distributed with your product:

RUNTIME.EXE	Executive - Distribute your customized version
INSTALL.TEM	Runtime Installation Launcher - Distribute your customized INSTALL.EXE version. You must write the INSTALL.PDM program which is launched by this program.
INSTALL.PDM	DeskMate application which installs your application onto a hard disk.
DMSETUP.ACC	Setup Accessory
DMSETUP.HLP	Setup Accessory Help File
DMCSR.R89	Core Services Resource
PRGUF.RES	Power & Run General User Functions Resource
DMMDJOY.RES	Tandy 1000 Joystick Driver
DMMDP.RES	Micro-Channel Serial Mouse Driver
DMMDSERI.RES	Serial Mouse Driver
DMVID.EXE	DeskMate video force utility.
DMVID.DOC	Video force utility documentation.

Distribute the video driver resolution set which is required by your application. If your application is a standard 80 column application, distribute ONLY these drivers:

DMVS1000.RES	Tandy 1000 (TGA), 4 color video driver
DMVSCGA.RES	CGA, 2 color video driver
DMVSEGA.RES	EGA, 16 color video driver
DMVSHERC.RES	Hercules, 2 color video driver
DMVSVGA.RES	VGA, 16 color video driver
DMVSTC16.RES	Tandy TL/SL (ETGA), 16 color video driver
DMVSMCGA.RES	MCGA, 2 color video driver

If your application is a 40 column application, distribute ONLY these drivers:

DMVSLRES.RES	40 column, low resolution video driver
DMVST256.RES	40 column, vga video driver
DMVSTC40.RES	40 column, Tandy 1000/TL/SL video driver
DMVSH.RES	40 column, Hercules video driver
DMVSE.RES	40 column, EGA video driver
DMVSM.RES	40 column, Monochrome EGA video driver

Files which must be distributed ONLY if your applications uses the specific function or resource:

DMPGSET.ACC	DeskMate Page Setup Accessory
DMPGSET.HLP	DeskMate Page Setup Accessory Help File
DMHELP.ACC	Help Accessory
DMHELP88.ACC	DeskMate 3.0 Compatible Help Accessory
DMHLPENG.RES	DeskMate Intelligent Help Resource
DMGUF.R89	General User Functions Resource
DMDB.R89	Database Control Resource, is required by the DeskMate Help System
DBBUILD.RES	Database File Build Resource
DBREAD.RES	Database File Read Resource, is required by DeskMate Help System
DBUPDATE.RES	Database File Update Resource
DMFORM.RES	Form Manager Resource
DMTHES.RES	Thesaurus resource (see local dealer).
DMPDASCI.RES	Daisy-wheel, or other non-supported printer, printer driver
DMPDIBMM.RES	IBM-compatible graphics printer driver
DMPD1.RES	Tandy DMP 105 printer driver (Tandy mode)
DMPD2.RES	Tandy DMP 200, 420, or 430 printer driver (Tandy mode)
DMPDLASR.RES	HP Laserjet Plus or Laserjet-compatible printer driver
PLAY.PDM	Play application, launches tutorial or demo
DMPLAY.RES	Play resource
DMUNPACK.RES	Tutorial Decompression Resource
DEMO.PDM	Customized Runtime Demo Launcher
TUTKBD.RES	Keyboard Layout Resource



Tandy Electronics

A DIVISION OF TANDY CORPORATION

RESEARCH AND DEVELOPMENT

1300 Two Tandy Center
Fort Worth, TX 76102
Telephone (817) 390-2181
Fax (817) 878-6575

April 20, 1990

Dear DeskMate Developer,

Tandy is pleased to let you know that the development of the DeskMate 03.04.00 product is well underway. LIM and shadow ram support for DeskMate applications and resources are among this year's product enhancements, both of which will be provided in the DeskMate retail and runtime products.

System requirements for using DeskMate in LIM are:

Expanded memory with an EMM driver (EMM.EXE, QEMM.EXE, etc.).

OR

Hardware expanded memory such as RamPage or AboveBoard.

Using DeskMate in Shadow RAM requires an XMS driver.

Tandy is providing this feature in the retail and runtime products to compliment the memory savings available on our DeskMate ROM machines. With the basic DeskMate environment in LIM, an additional 90K is available for your application. The basic environment includes DESK.EXE, DMCSR.R89, DMGUF.R89, and PRGUF.RES. With the environment in Shadow RAM as well as LIM, an additional 89K of system memory is available - total 179K. The amount of memory required by the XMS and/or EMM driver used is not included in this calculation and will reduce the total amount of memory available.

To ensure your DeskMate applications run successfully in this new configuration, we are making Beta versions of the product available for your testing needs. To receive your copy, simply complete the enclosed Order Form and return it to Tandy as soon as possible.

We also encourage developers to take advantage of the DeskMate LIM support in their own DeskMate applications and resources. Coding changes to your application or resource may not be required if your program follows these general guidelines.

- o The program does not use overlays.
- o The program can be split between its *pure segments*, code/data which does not change, and *impure segments*, code/data which does change.
- o The code segment(s) is 64K or smaller and does not contain interrupt routines or DeskMate event handlers. The code segment must also be at the beginning of the program.

If you are interested in supporting LIM in your DeskMate program, simply check the box labeled "LIM Support Technical Information" on the Order Form.

Thank you for your support of the DeskMate environment and development. We wish you continued success in your DeskMate development efforts.

Sincerely,

Rachel McKenzie
Manager, DeskMate Support Services

Enclosures

Missing (3)

3.5" version 3.05

disks, but Digital

Sound Toolkit is

unchanged.



Tandy Electronics

A DIVISION OF TANDY CORPORATION

RESEARCH AND DEVELOPMENT

1300 Two Tandy Center
Fort Worth, TX 76102
Telephone (817) 390-2181
Fax (817) 878-6575

April 19, 1990

Dear Software Developer,

This is your opportunity to join the growing number of software developers that are helping create an industry-standard user interface for MS-DOS software. This interface which is part of the DeskMate Development System, can help you create friendly, easy-to-use software.

Tandy is offering a two-day technical seminar, with a wide range of both introductory and advanced topics, to expand your knowledge of DeskMate and the DeskMate Development System. Come and find out just how easy it is to develop DeskMate applications.

Some of the companies currently using the DeskMate Development System include Lotus, Symantec, Logitech, and The Software Toolworks.

You should make plans to attend this seminar if you are a software developer:

- o currently developing DeskMate applications
- o interested in developing new applications using the DeskMate User Interface
- o interested in simply learning more about the DeskMate Interface

See the attached information and registration form to sign up!

Hope to see you in Fort Worth,

Terry Taylor
DeskMate Support Services

MR LABS

Attachment A: PC-LINK

Follow the steps outlined below to make use of the on-line technical support service.

- 1) Follow the steps outlined in your **PC-Link Connect Guide**, enclosed in your DeskMate 3 box, to familiarize yourself and register with the service. Contact PC-Link's Customer Service, informing them you are a DeskMate developer. This will speed up the approval process if you experience any delays.
- 2) Call the DeskMate Support Services hotline at 817-390-3664 to notify us that you are registered. Leave a message with your name, company name, and PC-Link screen name. We will contact PC-Link to give you access to the DeskMate Center.
- 3) Once you are cleared, use the "**deskhq**" keyword to get access to the DeskMate Center. Please register in the "Vendor Directory" area in Information Exchange so others can identify you by screen name.
- 4) Read the welcome message and the News from Tandy to familiarize yourself with the system.
- 5) Send confidential problems through E-Mail to TCTerryT or TCMikeH.
- 6) Post general problems in the Q&A sections.
- 7) Software updates are made through the Software Exchange and will be announced in News from Tandy.
- 8) Important messages will be broadcast by E-Mail to insure you receive them. The names appearing in the vendor directory will be sent the mail, be sure to register.



Tandy Electronics

A DIVISION OF TANDY CORPORATION

RESEARCH AND DEVELOPMENT
1300 Two Tandy Center
Fort Worth, TX 76102
Telephone (817) 390-2181
Fax (817) 878-6575

October 23, 1989

Dear DeskMate Developer,

Tandy provides on-line technical support, through **PC-Link**, for software developers writing DeskMate applications. The copy of DeskMate 3 which accompanied your development system includes a copy of PC-Link, the program used to access the service. The DeskMate Support Services group provides on-line support in the **DeskMate Center** forum through E-Mail, a Software Library for software updates, and message boards for problem reporting. We encourage all DeskMate developers to register and use this on-line service. Refer to attachment A for details about accessing PC-Link.

For software developers unable to use PC-Link, we recommend FAXing problem reports. Use the attached problem sheet to report problems by FAX. Scheduled software and documentation updates and news of importance will be made to you by regular mail. Please fill in and return the enclosed registration form to ensure the correct address information is used in our distribution list.

Fax Problem Reports to:
DeskMate Support Services
Attn: Problem Reports
817-390-2964

Tandy also provides telephone support, for assistance call 817-390-3664. Your call will be answered 24-hours a day by a voice-mail system. Leave a message, stating your name, company name, phone number, and the nature of your problem or question. Refer to Attachment B, the Problem Report Guidelines for suggestions on what information to include in your message. Your call will be returned between 9 a.m. and 5 p.m. CST and will be handled within one business day.

For timely and effective responses to your technical questions and problems, use one or more of the methods described above. We look forward to supporting you in the future in your DeskMate development efforts.

Sincerely,
DeskMate Support Services

Service and Solutions ...

... that's what the personal computer industry is all about. As a consultant or software developer, you're working daily to keep your clients "up and running," and still keep your business profitable.

Radio Shack's Consultant Liaison Program will help do that in more ways than one.

Tandy Computers: The broadest line of PCs in America. Radio Shack Computer Centers carry the full line of Tandy computers from our #1 selling PC-compatible Tandy 1000 family to our state-of-the-art Tandy 5000 MC.

We service what we sell: No more headaches from clients with "hardware" problems. Tandy Service Plans provide either on-site or carry-in service, freeing you from the hardware/service loop. Clients deal directly with Radio Shack factory trained representatives.

No "Information Overload": The Radio Shack CLP believes in providing you important information, like new product announcements, as quickly as possible. We'll also keep you up to speed with COM1: The Radio Shack Newsletter for Consultants. Published quarterly, a copy of this valuable newsletter is enclosed.

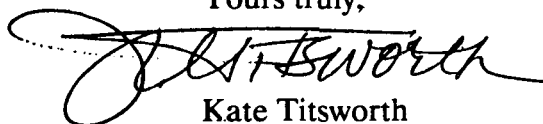
Nationwide Distribution Network: Over 290 Computer Centers nationwide, and an additional 400 Radio Shack PLUS Computer Centers dedicated to serving small and medium sized business, the home office and education.

Consultant Liaison Program members enjoy other benefits from the program including business and technical support, use of Computer Center facilities for consultant-sponsored seminars and a Finder's Fee Agreement.

Providing the best service and solutions for our customers has made Radio Shack a leader in our industry. Working together with industry specialists, like yourself, we make an unbeatable team.

Fill out and return the enclosed postage paid application. Or stop by a Radio Shack Computer Center for more information. Let's get started working together.

Yours truly,



Kate Titsworth
Manager
Consultant Liaison Program



Tandy Electronics

A DIVISION OF TANDY CORPORATION

RESEARCH AND DEVELOPMENT
1300 Two Tandy Center
Fort Worth, TX 76102
Telephone (817) 390-2181
Fax (817) 878-6575

To : DeskMate Developers
From : Rachel McKenzie
CC : Radio Shack, Customer Service
Subject : DeskMate 03.02.00 Addendum - Logitech Mouse Problem
Date : October 27, 1988

It has come to our attention that a bug in the Logitech mouse `click.exe` program causes the 03.02.00 versions of DeskMate and the runtime to lock up after displaying the copyright message. Tandy has found a way to work around the problem for users, with non-ROM machines, which require the program exist for their system to work. Users with the Tandy 1000 TL and SL models will have to remove the word "click" from their autoexec.bat file and may not use the command if they intend to run DeskMate.

We recommend you include the following information in your application documentation or contact your customer service group with the information so they can pass it on to your customers.

We recommend customers (and developer's) remove the word "click" from their `autoexec.bat` file. Those customers which require the click program should use the following procedure to fix their copies of `DESK.EXE` or `RUNTIME.EXE` :

Use the `patch.com` MS-DOS command to implement the following program changes to your DeskMate 03.02.00 executive.

PATCH `DESK.EXE,2B64,C1,C6` (retail version)
PATCH `DESK.EXE,2B7B,C1,C6`

or

PATCH `RUNTIME.EXE,2BB8,C1,C6` (runtime version)
PATCH `RUNTIME.EXE,2BCF,C1,C6`

Sincerely,

Rachel McKenzie
Rachel McKenzie
Senior Project Leader
DeskMate Vendor Support

cc LaDonna Womochel
Gene Schenberg
Dennis Tanner

Attachment B: Problem Report Guidelines

- 1) State the nature of the report
 - I have a question about DeskMate...
 - I am having a problem with my DeskMate application...
 - I need more information about the function call...
 - I have a suggestion to make...
 - I want to report a bug in DeskMate...
 - I would like to register a complaint...
- 2) If you need more information on a DeskMate function or function call, state the name of the function or the function you require and what information you desire.
- 3) If you think the problem you are encountering might be a bug in the DeskMate system, note the following:
 - a) Version/type of DeskMate environment exhibiting the problem, for instance DeskMate 03.00.00 or Runtime 03.03.01.
 - b) Date and model of DeskMate library linked with your application, for instance DM.LIB dated 6/15/89.
 - c) Whether or not you can duplicate the bug with a Tandy DeskMate application. If only your application exhibits the problem, note your application's memory requirements (code and data), memory model used, and whether or not its overlayed or packed.
 - d) Function call causing the bug and parameters passed to the call along with a duplication sequence.
 - e) Machine configurations on which bug occurs (DOS version, memory, video, any TSRs used, etc.)
- 4) Fill out the attached problem report, having your problem in writing will help when recording your message.

Dear DeskMate Developer,

This kit contains all of the necessary information to update your DeskMate Development System version 03.03.00 to version 03.05.00.

The items contained in this update kit are explained below.

1. DeskMate Development Guide Replacement Pages. These pages replace existing pages in your Development Guide. There is a completely new section About This Kit which explains in detail all of the new features of the DeskMate Development Kit version 03.05.00.
2. DeskMate Technical Reference Replacement Pages. There are two parts to this section. The first part, labeled 03.05 Replacement Pages, replace existing pages in your Technical Reference. The second part, labeled 03.05 Updates, is an update list. The Update lists minor changes to the Technical Reference. Replacement pages were not printed due to the number of pages required and the significance of the change .
3. Diskettes. All of the 03.05.00 Development Files are included. Also there are new SAMPLE.EXE and TOOLS.EXE to reflect the new sample programs and the updated development tools.

We have tried to make this Development Update Kit easy to understand and easy for you to assemble. As always we want to make your DeskMate Development effort as easy as possible.

Thank you for your continued support,

DeskMate Support Services

Volume I

DeskMate Development System
03.05.00 Updates

"About This Kit"

DeskMate Development Guide Replacement Pages

Changes to the DeskMate 3.03 Development Guide

About This Kit

This entire section is replaced by a new About This Kit.

DeskMate Development Guide

Title Page is replaced by new title page

Existing **Pages 1-9 thru 1-10** are replaced by new Pages 1-9 thru 1-10.

Existing **Pages 2-57 thru 2-64** are replaced by new Pages 2-57 thru 2-64.4.

Existing **Pages 3-17 thru 3-20** are replaced by new Pages 3-17 thru 3-20.

Existing **Pages 3-25 thru 3-26** are replaced by new Pages 3-25 thru 3-26.

Existing **Page 4-1** is replaced by new Page 4-1.

Existing **Pages 4-3 thru 4-9** are replaced by new Pages 4-3 thru 4-9.

Existing **Pages 6-3 thru 6-4** are replaced by new Pages 6-3 thru 6-4.

Existing **Pages 6-43 thru 6-44** are replaced by new Pages 6-43 thru 6-44.

Existing **Pages 6-65 thru 6-68** are replaced by new Pages 6-65 thru 6-68.

Existing **Pages 6-81 thru 6-82** are replaced by new Pages 6-81 thru 6-82.

Existing **Pages 6-105 thru 6-106** are replaced by new Pages 6-105 thru 6-106.

*These pages
have been
replaced*

DeskMate Development System
03.05.00
About This Kit

Contents

Chapter 1 - Introduction	1
Contents of the Kit	1
Using the Kit.....	5
 Chapter 2 System - Overview	 6
Technical Overview.....	6
DeskMate 3.05 Modifications and Enhancements.....	10
DeskMate 3.03 Modifications and Enhancements.....	10
 Chapter 3 - Getting Started	 12
Install DeskMate.....	12
Identify your Development System.....	12
 Chapter 4 - Registration and Technical Support	 14
PC-Link Information.....	15
Problem Report Guidelines.....	16
 Chapter 5 - The DeskMate Runtime License	 17
 Appendices	 18
A - Duplication License	
B - Registration Form	
C - Problem Report	

DeskMate Development System
03.05.00
DeskMate Development Guide

Chapter 1 Introduction

The DeskMate Development System (The Kit) allows software developers to write applications which run in a DeskMate environment. The DeskMate environment is provided by the DeskMate 3 product and runtime. This kit contains the documentation and software needed to develop a DeskMate application as well as a copy of the latest DeskMate 3 products. The DeskMate 3 products are included so programmers can acquaint themselves with the user interface, and execute and test their applications. The DeskMate 3.2 product should be used for compatibility testing. The DeskMate 3.03 product can be delivered up request. The runtime software is licensed and obtained separately, see Chapter 4 of this manual for more information.

The documentation in this kit is intended for experienced programmers who may or may not be experienced with DeskMate. Developing an application requires a proficiency in either the C programming language or assembly language on IBM PC-compatible machines. C and assembly language are the only languages Tandy currently supports for DeskMate development.

Contents of the Kit

The Kit includes two volumes of documentation. The first, DeskMate Development System 03.05.00 Development Guide contains three sections: About This Kit, DeskMate Style Guide, and the DeskMate Development Guide. The DeskMate Technical Reference is the reference for all function calls available in the environment and includes the Tandy Sound Toolkit I documentation and all of the new 3.05 information.

In addition to the documentation, the Kit includes three 3 1/2" diskettes which contain the development files, samples, tools, and utilities used in writing DeskMate applications and the Tandy Sound Toolkit I.

3 1/2" Package:

Disk #1 DeskMate Development System 03.05.00
Development Diskette

AUTOLOAD.H	Autoload Resource header file
CSRBASE.H	Header file for the Core Services Resource
CSRCFG.H	CSR Configuration header file
CSRCMPS.H	CSR Components header file
CSRFORM.H	CSR Form Manager header file
CSRKEYS.H	CSR Keyboard header file
CSRPR.T.H	CSR Print header file
CSRVID.H	CSR Video header file
DMDECL.H	DeskMate Function Prototype header file
DMDB.H	Database Resource header file
DMEXEC.H	Desk Executive header file
DMGUF.H	General User Functions Resources header file
DMTHES.H	Thesaurus header file
SPELL.H	Spell Checker header file
VENDOR.H	Vendor header file

AUTOLOAD.INC	Autoload Resource header file
CSRBASE.INC	Header file for the Core Services Resource

CSRCFG.INC	CSR Configuration header file
CSRCMPS.INC	CSR Components header file
CSRFORM.INC	CSR Form Manager header file
CSRKEYS.INC	CSR Keyboard header file
CSRPRNT.INC	CSR Print header file
CSRVID.INC	CSR Video header file
DMDB.INC	Database Resource header file
DMEEXEC.INC	Desk Executive header file
DMGUF.INC	General User Functions Resources header file
DMTHES.INC	Thesaurus header file
SPELL.INC	Spell Checker header file
THES.INC	Thesaurus Resource header file

DM.LIB	SMALL Module DeskMate Library
DMMED.LIB	MEDIUM Module DeskMate Library
	Note: Large model programs are not supported through the DeskMate library.

DESK.MAP	Map file for DESK.EXE used in DeskMate 3 product shipped
----------	--

Disk #2 DeskMate Development System 03.05.00
Samples, Tools, and Utilities

SAMPLES.EXE	Sample programs in a packed, self-extracting file
TOOLS.EXE	Tool and utility programs in a packed, self-extracting file

Disk #3 DeskMate Development System 03.05.00
Tandy Sound Toolkit I

DEMO1.EXE	Executable demo #1 which records a sound and plays it back.
DEMO1.C	Main source file for demo #1.
DEMO1.H	Main include declaration file for demo #1.
DEMO2.EXE	Executable demo #2 which records a sound, compresses it and stores it to disk.
DEMO2.C	Main source file for demo #2.
DEMO2.H	Main include declaration file for demo #2.
DEMO3.EXE	Executable demo #3 which reads a compressed sound from a file, decompresses it and plays it.
DEMO3.C	Main source file for demo #3.
DEMO3.H	Main include declaration file for demo #3.
DGETBUF.C	Allocates buffers for the SOUND and SNDHDR structures.
DSETUP.C	Initializes the buffers for the sound library.
DRECORD.C	Records a sound into allocated memory.
DPLAY.C	Plays back a sound from memory.
DSAVE.C	Takes a sound saved in the allocated buffers, compresses it and writes it to a file.

DLOAD.C	Decompresses a sound and loads it into the sound buffers.
SOUND.H	Main include file for use with the Tandy Digital Sound Library.
DSETUP.H	Include file for initialization.
DGETBUF.H	Include file for buffer allocation.
DRECORD.H	Include file for the record function.
DPLAY.H	Include file for the playback function.
DSAVE.H	Include file for the compress and save function.
DLOAD.H	Include file for the decompression and load function.
DEMO	Make file.
SOUND.LIB	Tandy Digital Sound Library.
BUILD.BAT	Batch file for compiling all 3 demos if make is not available.

The disk also contains a DeskMate sub-directory which contains a demonstration program utilizing the DeskMate interface.

DMDEMO.PDM	DeskMate executable which records a sound and plays it back, compresses and stores a sound to disk, reads a compressed sound from a file, decompresses it and plays it.
DMDEMO.C	Main source file for DeskMate demo.
DMGETBUF.C	DeskMate allocation of buffers for the SOUND and SNDHDR structures.
DMSETUP.C	DeskMate initialization of the buffers for the sound library.
DMRECORD.C	Records a sound into DeskMate allocated memory.
DMPLAY.C	Plays back a sound from memory utilizing DeskMate.
DMSAVE.C	Takes a sound saved in the DeskMate allocated buffers, compresses it and writes it to a file.
DMLOAD.C	Decompresses a DeskMate sound and loads it into the DeskMate sound buffers.
DMJOY.ASM	Sets and resets the Int 33 vector for the joystick.
DMDEMO.MKE	Make file for the DeskMate applications.
DEMOINFO.H	Contains all of the defines and structures for the DeskMate information boxes.

DEMOLDG.H Contains all of the defines and structures for the DeskMate dialog boxes.

All other *.H files are the same as the ones used in the non-DeskMate demos.

DMSEGS.INC Include file for DMJOY.ASM.

DMDEMO.LNK The link file which is called by the DeskMate make file.

DMDEMO.MAP The map file for the DMDEMO.PDM program.

SOUND.LIB Tandy Digital Sound Library.

Using the Kit

About This Kit, this manual, orients the programmer to the kit and how to use it. The System Overview which follows, describes the DeskMate operating environment and introduces most of the terminology used in DeskMate. It also highlights the changes made from the DeskMate 3.0 system through the current DeskMate 3.05 system. The Getting Started chapter discusses required development tools and installing the software. Registration, technical support, and the runtime license are discussed in Chapters 4 and 5. This manual should be read first since it answers many of the general questions you might have about DeskMate and developing for the environment.

The DeskMate Style Guide describes the DeskMate User Interface standards and guidelines that your application should follow. Conforming to the DeskMate standard is important since DeskMate users will expect your application to "look and feel" like the other DeskMate applications they already use. You should read this manual before and during the application design phase, especially while designing the application menus, dialog boxes, and screens. Review the guide again when the application is complete to ensure it meets the DeskMate standards.

The DeskMate Development Guide takes the programmer through the development life cycle of an application from getting started to distributing the software, including the development of help files, tutorials, and demos for the application. The manual includes an extensive set of programming examples that can be used as templates for your application. Also included in the guide is the documentation for the tools and utilities provided in the kit, including the help and tutorial tools.

The DeskMate Technical Reference is divided by functional entities and includes a description of every function call available to an application. The reference makes up the bulk of the documentation in the kit. The reference is divided by resources and managers within a resource. Refer to the Technical Overview which follows to identify a resource or manager.

The Development Diskettes contain the header files (*.H and *.INC) and libraries (DM.LIB and DMED.LIB) needed to build a DeskMate application. DeskMate does not support large model programs through these libraries. Refer to the DeskMate Development Guide, Getting Started section for a discussion of the implementation of large model programs under DeskMate.

The Samples and Tools Diskettes contain the sample programs and the tools and utilities supplied with the kit. The documentation for using these tools is also included in the DeskMate Development Guide.

The Tandy Sound Toolkit I diskette contains the header files, the Tandy Digital Sound Library (or simply sound library), and sample code (DeskMate and MS-DOS programs) needed to add sound to your application. The sound library includes a compression and decompression algorithm which produces results identical to that used in DeskMate 3 Sound and Music applications. For more information about the Sound Toolkit, see the Sound Library section of the DeskMate Technical Reference. Demo programs, for MS-DOS and DeskMate, are included on the Tandy Sound Toolkit I diskettes, for use as program templates.

Chapter 2 System Overview

DeskMate 3 was introduced in the fall of 1988. This version of DeskMate enabled developers to write applications for the interface and environment. This software is referred to as DeskMate 3.0 (or simply 3.0) and was not compatible with previous versions of the DeskMate product. DeskMate 3.0. DeskMate 3.03 was later released as an update to the 3.0 system. The following is a list of previously released versions:

1000 SL	DeskMate 03.00.00
1000 TL	DeskMate 03.00.00
Retail	DeskMate 03.00.00, 03.02.00, 03.03.01
Runtime	DeskMate 03.02.01, 3.03.01
1000 SL/2	DeskMate 03.03.00
1000 TL/2	DeskMate 03.03.00
1000 RL	DeskMate 03.04.00, 03.04.01

The latest DeskMate 3 product is version 3.05 which is compatible with earlier DeskMate 3 versions. DeskMate 3.05 includes the following versions:

Retail	DeskMate 03.05.00
Runtime	DeskMate 03.05.00

Technical Overview

The DeskMate 3 environment consists of the *executive* and the *resources* that contain the system functions. These resources provide the user interface, three levels of file input and output including a database, and printing support for the application.

DESK.EXE, the executive, loads and unloads DeskMate programs - *applications*, *accessories*, and *resources*, as well as non-DeskMate applications. The executive expects DeskMate programs to contain specific information in the program header used when the program is loaded. Refer to the DESKHDR.EXE documentation in the DeskMate Development Guide, Tools and Utilities section for more information.

DeskMate applications (.PDM extension) are the controlling modules in the environment.

Accessories (.ACC extension) are mini-applications which can pop-up over an application. Accessories are small programs that perform very specific tasks for the user and have functionality in several applications.

DeskMate resources (.RES extension) are the work-horses of the environment. They provide the common functionality required by most applications - user interface, file i/o, communications, printing, etc.. These resources are *terminate and stay-resident* (TSR) programs which have a focused scope of functionality. They may be shared by more than one program at a time. Resources are also used to provide device-specific functionality determined at execution time, such as, video and printer drivers.

All DeskMate programs must link with a DeskMate library to make use of functionality in the executive or any of the resources. The DeskMate libraries, DM.LIB and DMMED.LIB, contain the *bindings*, or bridge, used by a program to call a function in the resource.

When a program requires functionality provided by a resource, it *binds* to the resource by calling the executive and requesting the resource. The program binds to the resource once, before calling any functions in the resource. The bind call requests the executive to 1) find and load the resource and 2) inform the application of where the resource was loaded. When the executive loads the resource it resolves the resource's *service request vector* (srqv) in the bindings - the far address of the resource's entry point, and increments the resource's *use count*. The vector is used to make far calls from the application into the resource. The use count allows the executive to determine how many programs are using the same resource. When the program no longer requires the resource, it *freed* the resource. The executive decrements the resources' use count. The resource is not unloaded until all programs using the resource have freed the resource (its use count is zero) and the memory is needed to load another program.

The DeskMate 3.0 system used different file extensions to distinguish between the product resources (RES) and the runtime resources (RRS). Which resource was loaded is determined by the executive, DESK.EXE or customized RUNTIME.EXE, respectively.

The DeskMate 3.05 system uses the same file extension (RES) for both the product and the runtime resources. DeskMate system resources which must be used by the 3.05 system use the R89 extension to differentiate them from the 3.0 resources. The executive determines which resource to load.

These file conventions only affect developers who write their own resources. Resources developed for the 3.0 system have to be shipped with RES and RRS extensions for installation on a product or DeskMate 3.2 runtime. Resources developed for the DeskMate 3.05 system use the RES extension.

The executive, resources, and Setup, Page Setup, and Help accessories are distributed in the DeskMate runtime. These files are referred to as the *DeskMate system files*. Chapter 5 of this manual details how to obtain the runtime software.

The DeskMate executive and resources include:

DESK.EXE	The Executive handles the system level tasks for the environment and the applications. DESK Managers include: Clipboard Manager Desk Executive
DMDBBLD.RES	This Database Resource allows applications to create database files.
DMDBRD.RES	This Database Resource allows applications to open and read database files.
DMDBUPD.RES	This Database Resource allows applications to open, read, and update database files.

DMCSR.R89	<p>The Core Services Resource handles the user interface and "core" functionality for applications. This resource is always used by a DeskMate application.</p> <p>CSR Managers include:</p> <ul style="list-style-type: none"> Communication Manager Component Manager Configuration Manager Dialog Box Manager Event Manager Information Box Manager Keyboard Manager Menu bar Manager Message Box Manager Mouse Manager Print Managers Titleline Manager Video Manager Window Manager
DMDB.R89	This Database Resource provides database file access for applications developed with the 3.2 development system on a later system.
DMGUF.R89	<p>The General User Functions Resource provides high- and low-level file I/O functionality. See the Executive section for details on what functionality each GUF resource provides.</p> <p>GUF Managers include:</p> <ul style="list-style-type: none"> Environment Manager File I/O Manager
DMFORM.RES	The Form Manager Resource provides vector graphics and stroke font support for applications.
DMMD*.RES	The pointing device drivers J (Tandy 1000 joystick), S (serial mouse), P (Micro-Channel mouse)
DMPD*.RES	The printer drivers ASCI (ASCII), IBMM (IBM-compatible graphics), LASR (Laserjet-compatible), 1 (DMP 105), 2 (DMP 200, 420, 430), S (24-pin)
DMPD*.RFD	The resident font definition files needed for the video and printer drivers ASCI (ASCII), IBMM (IBM-compatible graphics), LASR (Laserjet-compatible), 1 (DMP 105), 2 (DMP 200, 420, 430), S (24-pin)
DMPE*.RES	The enhanced printer drivers ASCI (ASCII), IBMM (IBM-compatible graphics), LASR (Laserjet-compatible), 1 (DMP 105), 2 (DMP 200, 420, 430), S (24-pin)
DMVS*.RES	The 80 column video drivers CGA, 1000 (TGA), TC16 (ETGA), EGA, VGA, MCGA, and HERC. The 40 column video drivers LRES, TC40, T256, H (Hercules), E (EGA), and M (Monochrome EGA)

DMVE*.RES The 80 column enhanced video drivers CGA, 1000 (TGA), TC16 (ETGA), EGA, VGA, MCGA, and HERC.

PRGUF.RES The Power and Run General User Functions Resource, provides a subset of the GUF functionality. See the Executive section for details on what functionality each GUF resource provides.

The DeskMate libraries contain functionality which is linked into the application. The Library Manager lists all functions available in the libraries.

DeskMate 3.05 Modifications and Enhancements

The DeskMate 3.05 system added the following modifications or enhancements to the user interface and environment. For more information about how these changes affect your application's compatibility on all DeskMate 3 systems, refer to the DeskMate Development Guide, Getting Started section. For more information about the user interface changes refer to the DeskMate Style Guide. For more information about the enhanced calls, see the appropriate section in the DeskMate Technical Reference; the new calls are shown as "DeskMate 03.03 and later" in the **Special Notes** section of each call.

Environment

Font support has been provided to add new type faces through the use of the Form Manager.

Communications support now includes COM3 and COM4.

Extended memory support is now provided that adheres to the LIM standard.

Additional printer support is available, including the Epson 24-pin color printer.

DeskMate 3.03 Modifications and Enhancements

The DeskMate 3.03 system added the following modifications or enhancements to the user interface and environment. For information on the new utilities mentioned, refer to the DeskMate Development Guide, Tools and Utilities section.

User Interface

The title-line was rearranged to have the Help F1 prompter appear at the far left over the function key on the keyboard. The time indicator was moved to the far right.

In dialog boxes and message boxes, the default push button - button which will be pressed by the Enter key - appears with a dashed box around it to notify the user.

The "Sticky Menu" interface was added to the user interface. See the Development Guide, Getting Started section for more information if your application was predicting menu bar events.

Grayed menu items can be highlighted by the user, although they are not selectable (enhancement added for new help).

The busy icon is now animated, it cycles through a pattern.

An *About* menu option standard was adopted for all applications.

The "arrow algorithm" used in dialog boxes was optimized to eliminate "dead spots", components in a dialog box that could not be accessed via the arrow keys.

Environment

Applications written with the DeskMate 3.0 system used `SETHEAP.EXE` to set their minimum and maximum load size requirements. DeskMate 3.03 applications should use the new `DESKHDR.EXE` utility which also contains the program's split allocation, code shed size, and version number. The 3.03 executive will use the information stored in the new header when loading a program and when deciding how much code to shed when more space is needed to load an accessory. The 3.0 executive will ignore the extra information.

Additional accessories can now be accessed through the *More* option on the F10 Menu.

Context-sensitive help is now available in all pop-ups, dialog boxes, message boxes, and accessories, and on menu items.

The Form Manager which was part of the Core Services Resource, `DMCSR.RES`, was split out as a resource, `DMFORM.RES`. The CSR on a DeskMate 3.03 system will automatically load the resource on a `form_open` call for a 3.0 application.

The General User Functions Resource was split into two resources, `DMGUF.R89` and `PRGUF.RES`.

The Core Services Resource, CSR, only saves the first six (6) colors in the configuration file. Colors 7 through 16 are now considered to be application specific. The application must decide whether to save the colors with the data file or in its own configuration file.

The `dm_file_search` function honors diskette label files created with `DMLABEL.PDM`, a DeskMate utility.

The `dm_file_search` function no longer prompts the user to search the entire system before doing so.

The printer drivers were enhanced to support new line style widths, the patterns were changed to match the video drivers, the maximum number of characters printed on a line was increased, and several printing problems were fixed. For a complete list of changes, see the DeskMate Development Guide, Getting Started section.

A set of new video drivers was added, allowing applications to "video swap" into a 40 column screen resolution. These drivers require a DeskMate 3.03 system.

The Runtime executive now supports parameter passing to the runtime application and the execution of the runtime module from a DeskMate 3.0 Desktop. This feature is important to applications which require the DeskMate 3.03 system to operate.

Application Data Files

Data files for the DeskMate 3.05 product have the same format as was used in previous DeskMate 3 versions. The Address Book data file lengthened the Title and Address fields in DeskMate 3.03. Refer to Appendix A, DeskMate 3 File Formats in the DeskMate Development Guide for details.

Chapter 3 Getting Started

Install DeskMate

If you have not already done so, install the DeskMate product now. If you are not familiar with DeskMate, now is a good time to try out the product. The documentation in this kit often references the product when giving examples or explaining a concept.

Refer to the DeskMate Getting Started magazine for instructions on how to install and operate the software. DeskMate 3 operates on standard PC-compatible computer using MS-DOS 2.11 or later. The product requires a minimum of 384K of system memory to operate.

Identify your Development System

DeskMate applications are primarily written in C but may also be written in assembly language. The Kit does not contain the development tools necessary to write software, an editor, compiler, assembler, linker, or debugger, only those required to write a DeskMate application. Tandy recommends you use one of the following development systems for DeskMate development.

Compilers/Assemblers/Linkers

- Microsoft C 4.0, 5.0, or 5.1 with Microsoft MASM 5.0
- Microsoft C 6.0
- Microsoft Quick C
- Turbo C and Assembler 2.0

Debuggers

- Microsoft's SYMDEB from MASM 4.0
- Microsoft's CodeView
- Periscope
- Turbo Debugger

The kit supports small and medium model programs through its DeskMate libraries. It is possible to write a large model program by linking with the medium model library, `DMMED.LIB`, and managing your data model correctly. The Turbo C compiler requires modification to its startup code before it can be used for DeskMate development. Refer to the DeskMate Development Guide, Getting Started section for more information on both of these topics.

Install the DeskMate Development System

The Development Diskettes contain the header files and libraries needed to build an application. You may want to put the C header files (*.H) in your INCLUDE directory and the libraries in your LIB directory. If you are doing assembly language development, also place the header files (*.INC) in the INCLUDE directory.

The Samples and Tools Diskettes contain two packed files, SAMPLES.EXE and TOOLS.EXE. Each file is self-extracting and contains the sub-directory structure information within the file. To build the DeskMate samples and tools directories do the following:

```
>mkdir c:\desk\samples
>a:samples -d c:\desk\samples

>mkdir c:\desk\utility
>a:tools -d c:\desk\utility
```

The entire development system requires approximately 2 Meg of hard disk space for the development files, samples, and tools.

Chapter 4

Registration and Technical Support

Tandy provides on-line technical support, through PC-Link, for software developers writing DeskMate applications. The copy of DeskMate 3 which accompanied the kit includes a copy of PC-Link, the program used to access the service. The DeskMate Support Services group provides on-line support in the DeskMate Center forum through E-Mail, a Software Library for software updates, and message boards for problem reporting. We encourage all DeskMate developers to register and use this on-line service. Refer to the PC-Link information which follows for details about accessing PC-Link.

For software developers unable to use PC-Link, we recommend FAXing problem reports. Use the enclosed problem sheet to report problems by FAX. Scheduled software and documentation updates and news of importance will be made to you by regular mail. Please fill in and return the registration form to ensure the correct address information is used in our distribution list.

Fax Problem Reports to:
DeskMate Support Services
Attn: Problem Reports
817-390-2964

Tandy also provides telephone support, for assistance call 817-390-3664. Your call will be answered 24-hours a day by a voice-mail system. Refer to the Problem Report Guidelines which follow for suggestions on what information to include in your message. Your call will be returned between 9 a.m. and 5 p.m. CST and will be handled within one business day.

For timely and effective responses to your technical questions and problems, use one or more of the methods described above. We look forward to supporting you in the future in your DeskMate development efforts.

PC-Link Information

Follow the steps outlined below to make use of the on-line technical support service.

- 1) Follow the steps outlined in your *PC-Link Connect Guide*, enclosed in your DeskMate 3 box, to familiarize yourself and register with the service. Contact PC-Link's Customer Service at 800-458-8532, informing them you are a DeskMate developer. This will speed up the approval process if you experience any delays.
- 2) Call the DeskMate Support Services hotline at 817-390-3664 to notify us that you are registered. Leave a message with your name, company name, and PC-Link screen name. We will contact PC-Link to give you access to the DeskMate Center.
- 3) Once you are cleared, use the "deskhq" keyword to get access to the DeskMate Center. Please register in the "Vendor Directory" area in Information Exchange so others can identify you by screen name.
- 4) Read the welcome message and the News from Tandy to familiarize yourself with the system.
- 5) Send confidential problems through E-Mail to TCTerryT or TCBobT.
- 6) Post general problems in the Q&A sections.
- 7) Software updates are made through the Software Exchange and will be announced in News from Tandy.
- 8) Important information and announcements will be provided in the Information Exchange section.

Problem Report Guidelines

- 1) State the nature of the report
 - I have a question about DeskMate...
 - I am having a problem with my DeskMate application...
 - I need more information about the function call...
 - I have a suggestion to make...
 - I want to report a bug in DeskMate...
 - I would like to register a complaint...
- 2) If you need more information on a DeskMate function or function call, state the name of the function or the function you require and what information you desire.
- 3) If you think the problem you are encountering might be a bug in the DeskMate system, note the following:
 - a) Version/type of DeskMate environment exhibiting the problem, for instance DeskMate 03.00.00 or Runtime 03.05.00.
 - b) Date and model of DeskMate library linked with your application, for instance DM.LIB dated 6/15/89.
 - c) Whether or not you can duplicate the bug with a Tandy DeskMate application. If only your application exhibits the problem, note your application's memory requirements (code and data), memory model used, and whether or not its overlaid or packed.
 - d) Function call causing the bug and parameters passed to the call along with a duplication sequence.
 - e) Machine configurations on which bug occurs (DOS version, memory, video, any TSRs used, etc.)
- 4) Fill out the attached problem report, having your problem in writing will help when recording your message.

Chapter 5

The DeskMate Runtime License

The DeskMate Development System allows software developers to write DeskMate applications. These applications will run from the DeskMate DeskTop. To run these applications in a stand-alone environment (for customers who do not own the DeskMate 3 product), the software developer must use and distribute the DeskMate Runtime. To obtain a copy of the runtime software the software developer should sign and return the DeskMate Runtime Duplication License supplied in this kit. The license should be returned to

Dennis Tanner
Radio Shack
1600 One Tandy Center
Fort Worth, Texas 76102

Tandy will in turn sign the agreement and return a copy of the license to the developer along with a copy of the runtime software. The software is distributed in the 3 1/2" diskette format.

Refer to the DeskMate Development Guide, Distributing Your Application section for more information about how to distribute your DeskMate product.

Appendices

IMPORTANT NOTICE:

READ THE TERMS AND CONDITIONS OF THE LICENSE AGREEMENT BELOW CAREFULLY BEFORE OPENING THE SEALED DISK PACKAGE CONTAINING THE SOFTWARE. BY OPENING THE DISK PACKAGE YOU AGREE TO BE BOUND BY THE TERMS AND CONDITIONS, INCLUDING THE SOFTWARE LICENSE DISCLAIMER OF WARRANTIES AND LIMITATIONS OR LIABILITY CONTAINED THEREIN. IF YOU DO NOT AGREE TO THE TERMS AND CONDITIONS OF THE LICENSE AGREEMENT, YOU MUST RETURN THE PRODUCT WITH THE DISK PACKAGE UNOPENED TO THE PLACE OF PURCHASE WITHIN THREE (3) DAYS OF RECEIPT FOR A FULL REFUND.

TERMS AND CONDITIONS OF THE LICENSE AGREEMENT

I. SOFTWARE LICENSE

TANDY CORPORATION, its divisions and any associated subsidiary (hereinafter referred to as "LICENSOR"), grant to the original customer (hereinafter referred to as "LICENSEE") a non-exclusive paid-up license, to use the Software on one computer, subject to the following provisions.

- A. Except as otherwise provided in this Software License, applicable copyright law shall apply to the Software.
- B. Title to the medium on which the Software is recorded (cassette or diskette) or stored (ROM) is transferred to LICENSEE, but not title to the Software.
- C. LICENSEE may use the Software on a multiuser or network system only if the Software is expressly labeled to be for use on a multiuser or network system or if one copy of the Software is licensed for each node or terminal on which the Software is to be used simultaneously.
- D. LICENSEE agrees not to use, make, manufacture or produce copies of the Software except for use on one computer or as specifically provided in the Software License. LICENSEE is expressly prohibited from disassembling the Software.
- E. LICENSEE is permitted to make additional copies of the Software only for backup or archival purposes or if additional copies are required in the operation of one computer with the Software but only to the extent the Software allows a backup copy to be made.
- F. LICENSEE may transfer the Software to a third party provided that all original disks and documentation are included and the third party

agrees to be bound by the terms and conditions of this License Agreement.

- G. All copyright notices shall be retained on all copies of the Software.

II. LIMITED WARRANTY; OBLIGATIONS OF LICENSEE

- A. LICENSOR makes no warranty as to the design, capability, capacity or suitability for use of the Software, except as provided in this section. Software is licensed on an "AS IS" basis without warranty.
- B. For a period of Ninety (90) days from the date of receipt of the Software, LICENSOR warrants to LICENSEE that the Software is properly stored on the medium and that the medium itself is free from defects in materials and workmanship. This warranty is void if the Software has been subjected to improper or abnormal use. Defective Software must be returned to the place of purchase within the warranty period accompanied by a copy of the original receipt. LICENSEE's exclusive remedy, in the event of a Software manufacturing defect, is repair or replacement at LICENSOR's election.
- C. LICENSEE agrees to assume full responsibility that the Software meets the specifications, capacity, capabilities, versatility and other requirements of LICENSEE. LICENSEE agrees to assume full responsibility for the condition and effectiveness of the operating environment in which the Software is to function, and for its installation.
- D. LICENSEE agrees to perform backups of all data on a daily basis and assumes full responsibility for any loss of data or other consequences arising in whole or in part from the failure to do so.
- E. Except as provided herein, no employee, agent, dealer, distributor, or other person is authorized to give any warranties of any nature on behalf of LICENSOR.
- F. EXCEPT AS PROVIDED HEREIN, LICENSOR MAKES NO EXPRESS WARRANTIES, AND ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE IS LIMITED IN ITS DURATION TO THE DURATION OF THE WRITTEN LIMITED WARRANTIES SET FORTH HEREIN.

III. LIMITATION OF LIABILITY

- A. EXCEPT AS PROVIDED HEREIN, LICENSOR SHALL HAVE NO LIABILITY OR RESPONSIBILITY TO LICENSEE OR ANY OTHER PERSON OR ENTITY WITH RESPECT TO ANY LIABILITY, LOSS OR DAMAGE CAUSED OR ALLEGED TO BE CAUSED DIRECTLY OR INDIRECTLY BY THE SOFTWARE, INCLUDING, BUT NOT LIMITED TO, ANY

INTERRUPTION OF SERVICE, LOSS OF BUSINESS OR ANTICIPATORY PROFITS OR CONSEQUENTIAL DAMAGES RESULTING FROM THE USE OR OPERATION OF THE SOFTWARE. IN NO EVENT SHALL LICENSOR BE LIABLE FOR LOSS OF PROFITS, OR ANY INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY BREACH OF WARRANTY OR IN ANY MANNER ARISING OUT OF OR CONNECTED WITH THE LICENSE, USE OR ANTICIPATED USE OF THE SOFTWARE. NOTWITHSTANDING THE ABOVE LIMITATIONS AND WARRANTIES, LICENSOR'S LIABILITY HEREUNDER FOR DAMAGES INCURRED BY LICENSEE OR OTHERS SHALL NOT EXCEED THE AMOUNT PAID BY LICENSEE FOR THE LICENSE.

- B. No action arising out of any alleged breach of warranty, express or implied, may be brought more than one (1) year after the cause of action has accrued or more than two (2) years after the date of receipt of the Software, whichever first occurs.

IV. SEVERABILITY

Where a determination is made in any jurisdiction that any term or condition of this Agreement is invalid, unenforceable, illegal, or contrary to public policy, that provision shall be deleted but the remaining terms and conditions shall continue in full force and effect.

V. STATE LAW RIGHTS

- A. Some states do not allow limitations on how long an implied warranty may last, or the limitation or exclusion of incidental or consequential damages, so the above limitation(s) or exclusion(s) may not apply to LICENSEE.
- B. The warranties contained herein give the original LICENSEE specific legal rights, and the original LICENSEE may have other rights which vary from state to state.

Tandy DeskMate Runtime Version 03.03.01
Exhibit A

Executive Files and Accessories:

RUNTIME.EXE	Runtime Executive, Distribute Customized File
DMSETUP.ACC	Setup Accessory
DMPGSET.ACC	DeskMate Page Setup Accessory

DeskMate Help:

DMHELP.ACC	Help Accessory
DMHELP88.ACC	DeskMate 3.0 Compatible Help Accessory
DMHLPENG.RES	DeskMate Intelligent Help Resource
DMSETUP.HLP	Setup Accessory Help File
DMPGSET.HLP	DeskMate Page Setup Accessory Help File

Resources:

DMCSR.R89	Core Services Resource
DMGUF.R89	General User Functions Resource
PRGUF.RES	Power & Run General User Functions Resource
DMDB.R89	Database Control Resource required by DeskMate Help
DBBUILD.RES	Database File Build Resource
DBREAD.RES	Database File Read Resource required by DeskMate Help
DBUPDATE.RES	Database File Update Resource
DMFORM.RES	Form Manager Resource
DMTHES.RES	Thesaurus resource (displays see local dealer message).

Joystick and Mouse Drivers:

DMMDJOY.RES	Tandy 1000 Joystick Driver
DMMDP.RES	Micro-Channel Serial Mouse Driver
DMMDSERI.RES	Serial Mouse Driver

Printer Drivers:

DMPDASCI.RES	Daisy-wheel, or other non-supported printer, printer driver
DMPDIBMM.RES	IBM-compatible graphics printer driver
DMPD1.RES	Tandy DMP 105 printer driver (Tandy mode)
DMPD2.RES	Tandy DMP 200, 420, or 430 printer driver (Tandy mode)
DMPDLASR.RES	HP Laserjet Plus or Laserjet-compatible printer driver

Video Drivers:

DMVS1000.RES	Tandy 1000 (TGA), 4 color video driver
--------------	--

DMVSCGA.RES	CGA, 2 color video driver
DMVSEGA.RES	EGA, 16 color video driver
DMVSHERC.RES	Hercules, 2 color video driver
DMVSVGA.RES	VGA, 16 color video driver
DMVSTC16.RES	Tandy 1000 TL/SL (ETGA), 16 color video driver
DMVSMCGA.RES	MCGA, Multit-color video driver
DMVSLRES.RES	40 column, low resolution video driver
DMVST256.RES	40 column, VGA video driver
DMVSTC40.RES	40 column, Tandy 1000 TL/SL video driver
DMVSH.RES	40 column, Hercules video driver
DMVSE.RES	40 column, EGA video driver
DMVSM.RES	40 column, Monochrome EGA video driver

Tutorial and Demo Technology:

PLAY.PDM	Play application, launches tutorial or demo
DMPLAY.RES	Play resource
DMUNPACK.RES	Tutorial Decompression Resource
TUTKBD.RES	Keyboard Layout Resource
DEMO.PDM	Demo Launcher, Distribute Customized File

Others:

DMVID.EXE	DeskMate video force utility
DMVID.DOC	Video force utility documentation
INSTALL.TEM	Runtime Installation Launcher, Distribute Customized File
RUNTIME.MAP*	Runtime Executive Symbol Map File
RUNTMBLD.PDM*	Customize Runtime Utility
INSTLBLD.PDM*	Customize Installation Launcher Utility

* File is NOT for distribution.

142\5\88-51a.msg
11-151k

DeskMate Support Services Problem Report

Date Submitted: _____

Company Name: _____ Number: _____

Contact Name: _____ Fax: _____

Address: _____

Product/Project Name: _____

Circle One: Question / Suggestion / Bug Report / Complaint / Other

This image shows a single sheet of white paper with horizontal ruling lines. The lines are evenly spaced and run across the width of the page. There are no margins, text, or other markings on the paper.

DeskMate Style Guide

"DeskMate Style Guide"

Contents

Chapter 1 - Introduction

How to Use This Manual.....	2
Before You Begin	3

Chapter 2 - The Keyboard and the Mouse

Special Keys	5
Accelerators	5
Basic Mouse Operation.....	7
Cursors and Pointers	8
Cursors	8
Pointer Images	9
Selecting Data	9
Terminology	9
Selecting Graphics.....	11
Keyboard Interface.....	12
Mouse Interface	13
Scrolling	13
Keyboard interface	13
Mouse Interface	14
Cursor Movement and Scrolling.....	14
Arrow Keys.....	15
Home, End, Page Up, Page Down.....	15
Text Entry and Editing	17

Chapter 3 - Screen Design

General Guidelines	19
Parts of the DeskMate Screen	20
The Title Line	21
Help Prompter	21
Date	21
Application Name	22
Data File Name	22
Time	23
The Application or Accessory Menu Bar	23
The Work Area	24
Graphics	24
Windows	24
Monochrome and Color Support	25
Some Special-Purpose Screens	25
The Welcome Screen	25
The Default Screen	25
The Working Screen	26
Screen Design for 40-Column Applications	28

Chapter 4 - Menus and Menu Bars

What Are Menu Bars and Menus?	29
When to Use a Menu Bar	31
General Rules and Guidelines for Menu Bars	31
Menu Operation	31
Keyboard Interface	31
Mouse Interface	33
Menu Button Titles	33
Menu Option Names	34
Choosing and Using Menu Options	35
Accelerators and Selectors	35
Enabled and Disabled Options	37
Classes of Menu Options	37
Extended Command Options	37
Check Options	37
Menu Option Groups	38

Rules and Guidelines	
for Application Menu Bars	40
Usage and Location	40
Contents	40
On-Line Help	41
Exit, Run, and About	41
The Message Menu	42
The Accessories Menu	43
Rules and Guidelines	
for Accessory Menu Bars	44
Contents	44
Rules and Guidelines	
for Menu Bar Components	44
Usage and Location	45
Contents	45
Menu Bar Examples	46

Chapter 5 - The Interface Components

General Rules and Guidelines	49
Component Classes	50
Interactive Components	51
Check Boxes	51
When to Use Check Boxes	52
Rules and Guidelines for Check Boxes	53
List Boxes	54
When to Use List Boxes	54
Rules and Guidelines for List Boxes	54
Edit Fields	56
When to Use Edit Fields	56
Rules and Guidelines for Edit Fields	57
Edit Field/List Box Combinations	58
When to Use Edit Field/List Box Combinations	58
Rules and Guidelines for Edit Field/List Box Combinations	58

Push Buttons	59
When to Use Push Buttons.....	59
Rules and Guidelines for Push Buttons.....	59
Special Push Buttons - OK and CANCEL.....	61
Radio Buttons	61
When to Use Radio Buttons.....	62
Rules and Guidelines for Radio Buttons.....	62
Icon Buttons.....	64
When to Use Icon Buttons	65
Rules and Guidelines for Icon Buttons	65
Scroll Bars	65
When to Use Scroll Bars.....	65
Rules and Guidelines for Scroll Bars.....	66
Text.....	67
When to Use Text	67
Rules and Guidelines for Text	68
Icons.....	68
When to Use Icons	68
Rules and Guidelines for Icons	68
Boxes.....	68
When to Use Boxes.....	69
Rules and Guidelines for Boxes.....	69
Using Components in the Work Area.....	69
List Boxes	69
Edit Fields	70
Push Buttons	70
Icon Buttons.....	70

Chapter 6 - Pop-ups

When to Use a Pop-Up.....	71
Types of Pop-Ups	72
Message Boxes.....	73
Dialog Boxes.....	74
Accessories.....	75

Rules and Guidelines for Pop-Ups.....	75
Size and Position	75
Size and Position of Dialog Boxes	76
Size and Position of Message Boxes	76
Titles	76
Dialog Box Titles.....	77
Message Box Titles.....	77
Message Text.....	77
Pop-Up Operation.....	78
Default States.....	78
Disabled Components.....	78
Component Behavior	79
Push Buttons in Message Boxes.....	79
Assigning Default States.....	80
Removing Dialog Boxes.....	81
User Interfaces to Pop-Up Windows.....	81
Keyboard Interface	81
Mouse Interface	83

Chapter 7 - Special Menus

Message	85
Accessories.....	85
File	86
Edit.....	91

Chapter 1

Introduction

The Tandy DeskMate User Interface enables communication between users and DeskMate applications. It is a graphic interface that uses pop-up windows, pull-down menus, and a variety of other interface components to communicate with the user.

This manual provides the information you need to design and develop applications that meet the Tandy DeskMate User Interface standard. It presents rules (requirements) and guidelines (strong recommendations) for achieving the "look and feel" that users expect of DeskMate applications.

The main goal of the DeskMate standard is to help produce applications that are predictable and therefore easy to use. However, ease of use is only one benefit you receive from following the DeskMate standard.

The graphic nature of the DeskMate interface is a benefit to you and your users. You can use the graphic components to focus the user's attention where you want it. From the user's point of view, a graphic interface is appealing, not intimidating. It helps the user to feel comfortable with the application and confident about learning how to use it.

Using standardized components and interface techniques enables you to concentrate on improving your application, not on deciding how the interface components should operate.

The standard is flexible enough to support all the functionality you want in an application. It allows the freedom to implement unique features and functions. By following the standard, you can avoid forcing the user to learn a new interface with every DeskMate application.

How to Use This Manual

Read this manual before beginning development of a DeskMate application. It will help you select the interface components most appropriate for your application. If you have already started developing a DeskMate application, read this manual to ensure that your application conforms to the DeskMate standard.

This manual includes a large number of examples to help you understand the rules and guidelines in the standard. It includes subjects such as:

- screen design
- keyboard and mouse support
- how to use components in pop-up windows
- menus and menu bars
- special menus and functions
- special requirements for 40-column applications and accessories

This manual describes the look and feel of a DeskMate application. We recommend that you use DeskMate resources and functions in your application. If you create an application without using the DeskMate functions, the appearance and operation of your application should not violate any of the rules in this standard. For example, if your application uses pull-down menus, it is not essential that they all be sticky menus, but they must be consistent. Do not mix sticky menus and non-sticky menus.

If you use DeskMate functions to create and display the components your application uses, the application will automatically conform to a large portion of the standard.

Before You Begin

If you are not familiar with the DeskMate product, we recommend that you spend some time exploring the copy that is shipped with this kit. Understanding how to use DeskMate will help as you develop your application.

Most of the examples used in this manual are taken from the DeskMate product itself. If you are familiar with DeskMate, you will more quickly understand the context of the examples.

Chapter 2

The Keyboard and the Mouse

The keyboard is the standard input device for any DeskMate application. Some users prefer working with a mouse, however, so mouse support should be included whenever possible. All DeskMate applications should accept input from the keyboard, a mouse, or both.

This chapter describes the rules and guidelines that apply to using the keyboard or a mouse. It is organized according to the major tasks for which the mouse or keyboard is used. In addition, it includes a summary of basic mouse operation, and a summary of special keys and key combinations.

Special Keys

The keys and key combinations described in this section have special meaning in specific environments. Do not use these keys or key combinations for other purposes.

Accelerators

You can assign a combination of keystrokes, called an *accelerator*, to a particular function in your application. Accelerators provide quick access to menu options and other functions. Assign accelerators only to the most commonly used functions. Avoid accelerating every menu option, because the user cannot remember a large number of accelerators.

Esc

Esc, the Escape key, is the accelerator for the CANCEL button in dialog boxes and message boxes. It is also the accelerator for the Exit option in applications and accessories when a menu is not displayed. When a pull-down menu is displayed, Esc retracts the menu.

Enter

Enter is the accelerator for the currently highlighted push button in a dialog box.

Spacebar

Select objects such as a radio button, push button, and entry in a list box, or a graphic object.

Del

In a text entry window, Del deletes any selected text or the character to the right of the cursor. It can also be used as an accelerator. For example, in Text's Edit Menu, Del is the accelerator for the Clear option.

Function Keys

F1 is the accelerator for on-line help. F2 through F8 are accelerators to menu buttons. F9 is the accelerator for the Message Menu, and F10 is the accelerator for the Accessories Menu. Do not assign any other use to these keys.

Alt+key

You can use Alt+*key* combinations as accelerators. Alt+*first letter* is the standard accelerator for any push button. You can also use Alt with other keys as accelerators for menu options.

For example, in the Calendar accessory, Alt+P is the accelerator for the Prev (previous month) push button, and Alt+N is the accelerator for the Next (next month) push button.

You can also use Alt + special keys for cursor movement. For example, Alt+→ could move to the end of a word in a text entry window.

Ctrl+key

Ctrl+key combinations are menu option accelerators. For example, Ctrl+Ins selects the Copy option on the Edit Menu.

You can also use Ctrl + special keys for cursor movement. For example, Ctrl+→ could move to the end of a line in a text entry window.

Shift+key

You can use Shift with other keys as menu option accelerators. You can also use the Shift key with Home, End, Page Up, Page Down or the arrow keys to select text or graphics in the work area. For example, Shift+Up Arrow is used to select items in a multi-select list box. See the "Selecting Data" section of this manual for specific key combination standards.

Shift+key is often related to the function of key. For example, Shift+Del selects the Cut option on the Edit Menu. This deletes currently selected text from the editing window and transfers it to the clipboard. Shift+Ins selects the Paste option on the Edit Menu. This inserts text from the clipboard into the current text window, at the current cursor location.

Shift+key may reverse the action of key. For example, Tab moves the highlight forward in a window; Shift+Tab moves the highlight backward in a window.

Basic Mouse Operation

We use the term mouse to refer to a pointing device. A pointing device can be either a mouse or a joystick. Using a mouse can expedite data selection and scrolling.

The DeskMate mouse interface requires only one button. If your mouse has more than one button, use the left one to interact with DeskMate applications. This section describes the basic actions of a mouse.

Click

Press and quickly release the mouse button once. This action re-positions the text cursor or selects an object.

Double Click

Click the mouse button twice in rapid succession. Used to select and immediately execute an option.

Shift Click

Press the Shift key and the mouse button at the same time. Used to make discontinuous selections.

Drag

Press the mouse button and hold it down while moving the mouse. Release the mouse button when the pointer rests at the desired location. Used to make contiguous selections.

Cursors and Pointers

The cursor indicates the point where activity will occur in a DeskMate application. For example, in a text application, the cursor indicates the location at which text can be inserted, deleted, or changed.

The pointer indicates the presence of the mouse. When you move the mouse, the pointer moves. When you click the mouse, the cursor moves to the position of the mouse pointer.

Cursors

Standard DeskMate functions provide three predefined text cursors: a block, a bar, and a line. If you prefer, you can define a custom cursor and use it instead.

- Use the block cursor or the bar cursor in a text entry window.
- Use the line cursor to focus on a component or an application-defined object.

Pointer Images

DeskMate defines a standard mouse pointer image that is available for use by all applications. An application can define other images for the mouse pointer and use them as long as the application is running. When the user exits the application, it must redefine the mouse cursor to the standard DeskMate mouse cursor.

Selecting Data

The DeskMate user interface is action-oriented. The user selects data, then performs some action on that data. When data is not selected, the actions in the applications should be disabled.

DeskMate applications can support selection of text, graphics, or any other kind of data that is appropriate for the application.

If an application supports the selection of any kind of data, it must support the clipboard. The clipboard is a temporary storage area for data. Your application can use the clipboard to move data from one location in a file to a different location in the same file, or to a different file.

Terminology

Some important terms that relate to text selection are defined below. These terms apply whether you use the mouse interface or the keyboard interface. Understanding these terms will help you understand the mouse and keyboard descriptions later in the chapter.

Anchor Point

The point where an extended text selection begins. The anchor point must be the top or bottom of the selected text; it cannot be in the middle of the selected text. The user makes extended text selection with Shift+key combinations on the keyboard, or by dragging the mouse. For the keyboard, the

anchor point is the position of the cursor at the time the first Shift+key combination is invoked. For the mouse, the anchor point is the position of the first character selected when the user begins the drag operation.

Character

Any single character, including a space or a punctuation mark.

Word

A group of characters separated from other characters on either side by a space, a tab character, or a new-line character.

Line

One row of characters with application-defined boundaries on each end. For example, the characters within an edit field frame are a line, according to this definition. The screen border can also serve as a line boundary.

Extended selection

Expansion of the current selection to include additional characters, words, or lines.

Discontiguous selection

Selection of two or more items that are not adjacent.

Deselected data

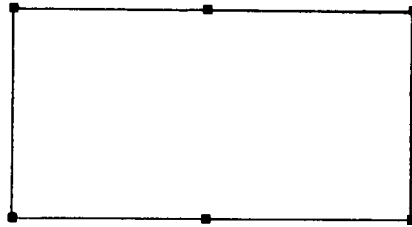
A selection is deselected when the user moves the cursor without extending or contracting the current selection.

Whenever possible, the application should deselect data for the user, rather than requiring the user to do so. For example, after executing the Copy option to store the information on the clipboard, the application should deselect the data and position the cursor at the anchor point.

The application should position the cursor at the anchor point after completing an operation involving the data.

Selecting Graphics

When a graphic object is selected, it appears in a select box:



The small squares around the perimeter of the select box are size handles. To change the size of a graphic object, the user clicks and drags any one of the size handles. The shape of the object changes as the user drags the handle. To move a graphic object, the user clicks and drags the object itself.

If your application allows selection of graphics, it must support the following keys:

Arrow Keys

Use the arrow keys to position the cursor on the graphics or the handles of an object you want to select.

Space bar

The space bar highlights the graphic element at the current cursor location, and selects the handle.

Your application should also support graphic selection by the mouse. The user can click on any handle to select graphics with the mouse.

Keyboard Interface

The key combinations described in this section move the cursor and select text.

Shift+Up Arrow

Moves the cursor up one line and scrolls if necessary.
Selects/deselects data from the original cursor location to the new cursor location.

Shift+Down Arrow

Moves the cursor down one line and scrolls if necessary.
Selects/deselects all text from the current cursor location to the new cursor location.

Shift+Right Arrow

Moves the cursor to the right one character and scrolls if necessary. Selects/deselects the character. At the end of a line, the cursor moves down one line and to the left edge of the new line.

Shift+Left Arrow

Moves the cursor to the left one character and scrolls if necessary. Selects/deselects the character. At the end of a line, the cursor moves up one line and to the right edge of the new line.

Shift+Home

Selects all text from the current cursor location to the beginning of the current line and scrolls if necessary.

Shift+End

Selects from the current cursor location to the end of the current line and scrolls if necessary.

Shift+Ctrl+Home

Selects from the current cursor location to the beginning of the file and scrolls if necessary.

Shift+Ctrl+End

Selects from the current cursor location to the end of the file and scrolls if necessary.

Shift+Page Up

Selects/deselects text and scrolls if necessary. The cursor positioning and scrolling rules for Page Up apply.

Shift+Page Down

Selects/deselects text and scrolls if necessary. The cursor positioning and scrolling rules for Page Down apply.

Shift+Ctrl+Page Up

Selects/deselects text and scrolls if necessary. The cursor positioning and scrolling rules for Ctrl+Page Up apply.

Shift+Ctrl+Page Down

Selects/deselects text and scrolls if necessary. The scrolling rules for Ctrl+Page Down apply here.

Mouse Interface

Pointing, pressing the mouse button, and then dragging the mouse selects or deselects text or graphics. At window boundaries, scrolling occurs as necessary.

Scrolling

When data does not fit in the available space on the screen or in a window, the application must support scrolling. This section describes the keys and mouse actions used while scrolling.

Keyboard interface

DeskMate uses a variety of keys and key combinations to support scrolling through a file. Refer to "Selecting Data" for information about the effects of these keys on cursor positioning and scrolling.

Mouse Interface

The mouse supports scrolling with arrow buttons on the menu bar or a scroll bar component. This section describes mouse operations that are equivalent to pressing the keys listed in the last section.

- Clicking on an arrow button in a menu bar or on a scroll bar is equivalent to pressing the corresponding arrow key.
- Clicking in the gray area above or below the vertical scroll bar elevator is equivalent to pressing the Page Up or Page Down key.
- Clicking in the gray area to the left or right of the horizontal scroll bar elevator is equivalent to pressing the Ctrl+Page Up or Ctrl+Page Down key combination.
- Dragging the scroll bar elevator moves the cursor quickly through the file. The cursor the same relative distance from the top of the file as the elevator is from the top of the gray area in the scroll bar.

Cursor Movement and Scrolling

DeskMate uses a variety of keys to move the cursor on the screen. You can also use most of the keys in this section to select data. Refer to "Selecting Data" for more information.

Tab

Moves the highlight (cursor) forward to the next component or field.

Shift+Tab

Moves the highlight backward to the next component or field.

Arrow Keys

The arrow keys (Up, Down, Left, and Right) provide rapid motion through a file. You can use these keys as described in this section.

Up Arrow

Moves the text cursor directly up one line. If the cursor is at the top of a window that scrolls up and down, pressing the up arrow scrolls the text down one line without moving the cursor.

Down Arrow

Moves the text cursor directly down one line. If the cursor is at the bottom of a window that scrolls up and down, pressing the down arrow scrolls the text up one line without moving the cursor.

Right Arrow

Moves the text cursor right one character on the same line.
Scrolls the text to the right if the cursor is at the far right of a side-scrolling window.

Left Arrow

Moves the text cursor left one character on the same line.
Scrolls the text if at the far left of a side-scrolling window.

Home, End, Page Up, Page Down

You can use Home, End, Page Up, and Page Down alone, or paired with the Ctrl key to position the cursor. This section describes the effect of each of these keys.

Home

Moves the text cursor to the beginning of the current line and scrolls the text if necessary.

End

Moves the text cursor to the end of the current line and scrolls the text if necessary.

Ctrl+Home

Moves the text cursor to the first character in the first line of the file (also called *homing* the cursor).

Ctrl+End

Moves the text cursor to the last character in the last line of the file.

Page Up

Re-positions the text cursor and scrolls the text if necessary. When the cursor is not on the top line of the screen, the cursor moves to the top of the current screen. When the cursor is already at the top of the screen, it moves one screen toward the beginning of the file each time the user presses the key. The top line of each screen becomes the last line of the next screen, so the user always sees one line from the previous screen of text.

Page Down

Re-positions the text cursor and scrolls the text if necessary. When the cursor is not at the bottom of the screen, the cursor moves to the bottom of the current screen. When the cursor is already at the bottom of the screen, it moves one screen toward the end of the file each time the user presses the key. The bottom line of each screen becomes the first line of the next screen, so the user always sees one line from the previous screen of text.

Ctrl+Page Up

Re-positions the text cursor and scrolls the text if necessary. When the cursor is not at the left edge of the screen, the cursor moves to the far left edge of the screen, in the current line of text. When the cursor is already at the left edge of the screen, it moves one screen toward the left edge of the file each time the user presses the key sequence. The first column of one screen appears as the last column of the next screen, so the user always sees one column from the previous page of text.

Ctrl+Page Down

Re-positions the text cursor and scrolls the text if necessary. When the cursor is not at the right edge of the screen, the cursor moves to the right edge of the screen, in the current line of text. When the cursor is already at the right edge of the screen, it moves one screen toward the right edge of the file each time the user presses the key sequence. The last column of one screen appears as the first column of the next screen, so the user always sees one column from the previous page of text.

Text Entry and Editing

Entering and editing text is largely a keyboard activity. This section describes the keys that DeskMate uses to support text entry and editing.

Del

Deletes any selected text or the character to the right of the text cursor.

Enter

Inserts a line feed and a carriage return, which moves the cursor to the left margin. If text is selected, it is replaced with a line feed and carriage return.

Space bar

Inserts a space in the text. If text is selected, it is deleted and replaced with a space.

Tab

Moves the text cursor forward one tab stop.

Shift+Tab

Moves the text cursor back one tab stop.

Chapter 3

Screen Design

This chapter describes the major areas of the screen and the rules and guidelines that apply to each area. In addition, it defines a few special types of screens used in many DeskMate applications, and includes a section on 40-column applications.

General Guidelines

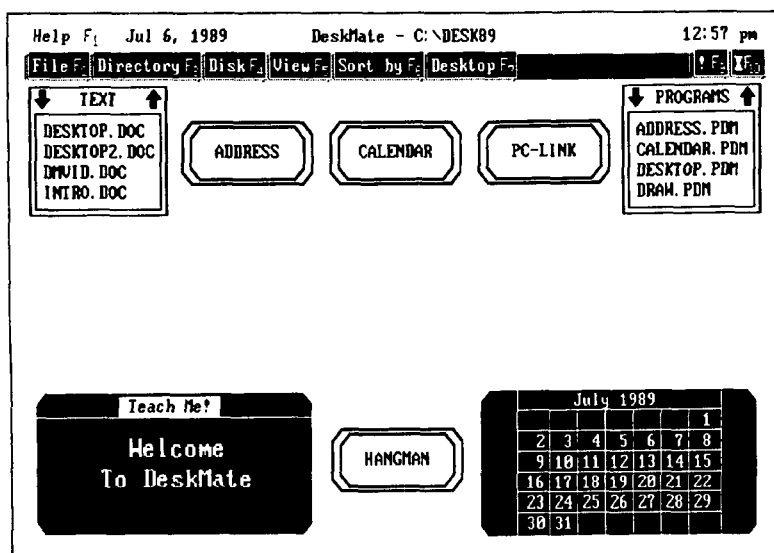
Screen design is a broad topic, and the guidelines are correspondingly broad:

- Keep the screen organized to minimize confusion.
- Use screen areas consistently to maintain familiarity and help the user locate information quickly. For instance, try to position the OK and CANCEL buttons in the same place in every dialog box.
- Use dialog and message boxes effectively. Be sure the dialog is organized and contains all the information the user needs. Too much information at once will confuse users; a lack of needed information will frustrate them.

Parts of the DeskMate Screen

The DeskMate screen is divided into three major parts:

- The *title line*, the first line of the application screen
- The *application menu bar* or *accessory menu bar*, which appears immediately below the title line
- The *work area*, the space below the menu bar, in which all the application's functions are performed



Each of these parts serves a specific purpose, and each has its own set of design rules and guidelines.

The Title Line

All applications must include a title line at the top of the main screen. The title line contains:

- the Help prompter
- the current date
- the name of the application
- the name of the current data file
- the current time

If you use DeskMate functions to display your title line, the title line will automatically be formatted to conform with the rules in this section.

Help Prompter

A *prompter* is a display of a particular task's accelerator in a DeskMate application. F1 always provides on-line help; the prompter is Help F1. Display this prompter at the left end of the title line.

Date

Display the current date as the second element in the title line. The date appears in the format *mmm dd, yyyy*, as in Aug 24, 1989.

Application Name

The application name is the third element in the title line.

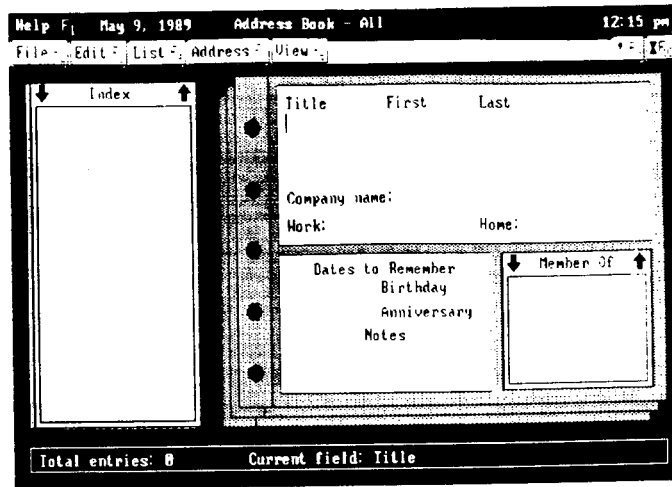
- Use a maximum of 16 characters for the name.
- Center the name horizontally.
- Capitalize the first letter of each word (for example, Text, Worksheet, Address Book, Form Setup).
- Insert a hyphen (-) between the application name and the data file name (described next).

Data File Name

If an application or accessory uses a data file, display the data file name to the right of the application name.

- Use uppercase characters, such as EXAMPLE.DOC.
- If the drive, path, and data file name exceed 28 characters, display only the drive and data file name, as in C:\...\EXAMPLE.DOC.
- If the application or accessory does not use a data file, display Untitled in place of a data file name.
- Some applications, such as Address Book, use only one data file. When this is the case, the application can use the data file name location for some other purpose.

For instance, Address Book uses this location to display the name of the current address list.



Time

Display the current time as the last element in the title line, at the right end of the title line. The time appears in the format hh:mm, followed by am or pm.

The Application or Accessory Menu Bar

All full-screen applications must include an application menu bar immediately below the application's title line. This menu bar must provide access to application functions, the DeskMate Accessories Menu and the DeskMate Message Menu. See Chapter 4, "Menu Bars and Menus," for detailed rules and guidelines on the application menu bar.

The title line and menu bar usually appear together. If an accessory uses a title line, it must include an accessory menu bar. However, an accessory that uses an accessory menu bar is not necessarily required to use a title line. See Chapter 4, "Menus and Menu Bars," for details.

The Work Area

The application work area is immediately below the menu bar. Developers can use graphics, windows, and color to focus the user's attention on the current task. Any DeskMate interface component can be used in the work area.

The rules that apply to the work area depend on how you use the area. See Chapter 2 if you are using the work area for data selection. See Chapter 5 if you are using interface components in the work area. Some guidelines are provided here.

Graphics

Graphics can be used to make the work area look like a familiar object. For example, the Address Book work area looks like a page from an address book. Users know automatically what to do with the form; little explanation is needed.

We realize this is not as simple in all applications as it is in Address Book. In general, simpler screen designs are easier to use. Make your screens as recognizable and usable as possible.

Windows

Windows can be used to concentrate the user's attention on a specific action. In DeskMate applications, windows can be used to get information from the user or just to send a message to the user. DeskMate applications use *tiled windows* and *pop-up windows*. Both types of windows are described in Chapter 6.

Monochrome and Color Support

DeskMate runs in a variety of video, color, and resolution modes. DeskMate applications must support monochrome (2-color), 4-color, and 16-color resolution environments.

Some Special-Purpose Screens

The following sections describe three special screens that help create the look and feel users expect in a DeskMate application:

- The welcome screen
- The default screen
- The working screen

The Welcome Screen

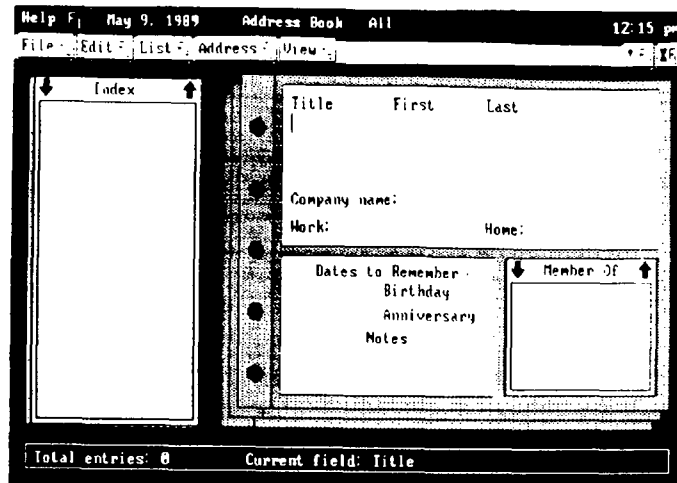
Use a welcome screen to convey any information that the user should read before using the application. Copyright information, which can also be displayed with the About option, is an example.

The Default Screen

Applications should have a default screen that is displayed as soon as the user starts the application.

Create a default screen that represents the function of your program. For instance, the Address Book screen looks like an address book entry page with an index pad on the side.

Data entry, editing, and viewing are all done on this screen.



The Working Screen

If your application includes more than one "mode" of operation, create a working screen for each mode. Working screens should visually identify the current mode of operation. For example, Calendar has daily, weekly, monthly, and yearly viewing modes. The current mode is easily identified by the screen's design.

The following examples illustrate the monthly mode and a daily schedule planner.

Help F1 Jul 6, 1989 Calendar - CONFERENCE RM 1:07 pm

File F2 Edit F3 Display F4 Search F5 Event F6 Calendars F7

PREV July 1989 NEXT

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
						1
2	3	4 ♦	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

Help F1 Jul 6, 1989 Calendar - CONFERENCE RM 1:18 pm

File F2 Display F4 Search F5 Event F6 Calendars F7

PREV Schedule for Friday July 7, 1989 NEXT

8:00 AM - 5:30 PM	DeskMate Seminar
12:00 PM - 1:00 PM	Lunch
6:00 PM -	Depart for home

Screen Design for 40-Column Applications

Screen design rules for 40-column applications are the same as for full-screen applications, with the following exceptions:

- Include only the Help prompter, the application name, and the current time on the title line. do not include the current date or the data file name.
- Do not include the Accessories Menu or the Message Menu on the accessory menu bar.

Chapter 4

Menu Bars and Menus

This chapter introduces menu bars and menus and defines the requirements of each of DeskMate's classes of menu bars. In addition, it discusses the appearance, design, operation, and interface support required for each class of menu bar.

In addition, this section discusses the rules and guidelines that apply to menu options, and discusses the types of commands that can be included in a menu.

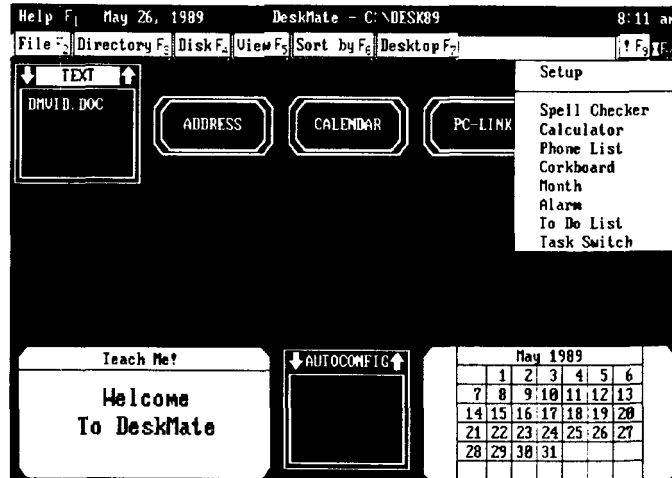
What Are Menu Bars and Menus?

A *menu bar* is a group of rectangular buttons, always displayed horizontally across the screen. Each button has a title, and each button activates (pulls down) a *menu* that has the same title. A menu is a group of related commands or settings. Each entry in a menu is called a *menu option*.

For DeskMate versions 3.03 and higher, all DeskMate menus are *sticky menus*. This means that a menu, once selected, remains displayed until the user selects an option from that menu, selects another menu, or cancels the current operation. If the user selects another menu, the first menu is erased from the screen and the second one is displayed instead.

DeskMate uses three types of menu bars. The first type is called the *application menu bar*. Each DeskMate application has a unique application menu bar.

The following illustration shows DeskMate's application menu bar. The Accessories Menu is pulled down.



The application menu bar allows access to all application menus, and the DeskMate menus.

The second type of menu bar is the accessory menu bar. An accessory menu bar serves the same purpose in an accessory that the application menu bar serves in an application.

The third type of menu bar is a *menu bar component*. Any menu bar that is included in a pop-up window, such as a dialog box, is a menu bar component.

When to Use a Menu Bar

Use a menu bar to present groups of related commands to the user. The menu bar interface is preferred over a series of push buttons because:

- It is the standard interface for presenting commands (actions) to the user.
- It is easier to access with the keyboard than a series of buttons, which must be cycled through or accelerated.

General Rules and Guidelines for Menu Bars

The rules and guidelines in this section apply to all menu bars. Rules and guidelines that apply specifically to a particular type of menu bar component are presented later in this chapter.

Menu Operation

Applications must provide access to menu options through the keyboard or the mouse. DeskMate functions support both interface devices as described in this section. The descriptions given here are general. Specific requirements for the various types of menu bars are given later in this chapter.

Keyboard Interface

To display an application menu, the user must press the appropriate function key, F2 through F10. If an application menu is already displayed, the user can retract it and display another by pressing a different function key or the left or right arrow key. The Esc key retracts a displayed menu without displaying another.

Function keys enable the user to display a menu from the menu bar. The Enter key, the alphabetic keys, and the up and down

arrow keys allow the user to highlight an option from the current menu.

A brief description of each major part of the keyboard interface follows:

F2 through F10

Pulls down a menu. Retracts the current menu if one is displayed.

Esc (Escape)

Retracts the pull-down menu without displaying another.

Right Arrow

Retracts the current menu and displays the one on its right. If the current menu is on the right end of the menu bar, displays the menu on the left end.

Left Arrow

Retracts the current menu and displays the one on the left. If the current menu is on the left end of the menu bar, displays the menu on the right end.

Enter

Executes the highlighted option and retracts the menu.

A through Z

Highlights the first option in the current menu that begins with the pressed letter. For example, pressing P while in the Edit Menu highlights Paste.

Up Arrow

Moves the highlight up one option in the current menu. If the first option in the menu is highlighted, moves the highlight to the last option in the menu.

Down Arrow

Moves the highlight down one option in the current menu. If the last option in the menu is highlighted, moves the highlight to the first option in the menu.

Mouse Interface

To display an application menu, the user must click on the appropriate menu button, F2 through F10. If a menu is already displayed, the user can click on a different button to retract the menu and display another.

If the application supports scrolling and does not use a scroll bar, the application menu bar must support arrow buttons. Menu bar components must not support arrow buttons.

A brief description of supported mouse actions follows:

Clicking on a menu button

Pulls down the corresponding menu. This is equivalent to pressing a function key.

Clicking a menu button and dragging the mouse to an option

Selects and executes the option, and retracts the menu. This is equivalent to pressing a function key, a series of down arrows, and the Enter key.

Double clicking on a menu option

Selects and executes the menu option, and retracts the menu. This is equivalent to pressing the Enter key.

Clicking on a point that is not in the menu or menu bar

Retracts the current menu without displaying a new one. This is equivalent to pressing the Esc key.

Menu Button Titles

Each menu button must have a title, or name, stating its purpose as clearly as possible. When titling a menu button:

- Make short titles. Use a single word or a short phrase.
- The title can be a noun (such as *File*, *Text*, or *Picture*) or a verb (such as *Edit*, *Search*, or *Zoom*).

- Do not include numbers.
- Do not pad the titles with spaces.
- Capitalize the first letter of the title.

Menu Option Names

- Menu options should always relate to the menu title.

For example, the Page setup, Save, and Save options in the File Menu all relate to the current application data file, as the word *File* implies.

- Menu option names must be unique within a menu; however, the same name can be used in different menus.
- If the menu name is a noun, the menu option names must be verbs, for example:

File Menu options are Open, Save, and Merge.

Text Menu options are Bold and Underline.

Picture Menu options are Move, Size, Hide, and Show.

- If the menu name is a verb, the menu option names must be verbs or adjectives, for example:

Edit Menu options are Cut, Copy, and Paste.

Search Menu options are Find, Find next, and Substitute.

Zoom Menu options are Normal, Twice, and All.

- If the menu name is a phrase, the menu option names should complete the phrase. For example:

Sort by Menu options are Date, Type, and Name.

- Spell out option names in full, and capitalize the first letter of the first word.

Examples of proper option names are: New, Open, Page setup, Save as, and Select all.

- Do not repeat the name of the menu in the menu option.

For example, if the menu name is File, the menu option should be Open, not Open file or File open.

Choosing and Using Menu Options

The DeskMate User Interface provides quick, easy access to application functions and data. This section describes some of the features that provide easy access to menu options.

Accelerators and Selectors

Accelerators provide quick keyboard access to menu buttons or menu options. Menu button accelerators are function keys. Menu option accelerators are usually a sequence of keys, although a single key can be used in some instances. Pressing a menu button accelerator displays the associated menu. Entering a menu option accelerator is equivalent to displaying the menu, choosing the option, and pressing Enter.

Selectors also provide access to menu options. Selectors are single keys, usually the first letter of the option name. A selector highlights an option, but the user still must press Enter to start the option.

Accelerators are required for all buttons on a menu bar. Display the accelerator to the right of the menu name on each menu button. If you use DeskMate functions to create and display your menu bar, menu button accelerators will be assigned and displayed automatically.

Accelerators are not required for menu options. If you assign accelerators to menu options, display each accelerator sequence to the right of the appropriate option name. The following example illustrates the use of accelerators in a menu.

Records	F4
First	Ctrl+F
Next	Ctrl+N
Previous	Ctrl+P
Last	Ctrl+L

The displayed accelerator sequence is called the prompter.

- Capitalize the first letter of each word in the prompter.
- Use a plus sign to connect the words in a prompter. Do not insert spaces between the plus sign and the words.
- Use Shift to indicate the Shift key, Ctrl to indicate the Ctrl key, and Alt to indicate the Alt key.
- All prompters must begin one character beyond the longest option name on the menu. If you use standard DeskMate functions, DeskMate can automatically format the menu as required by this standard. The following example, Calendar's Edit Menu, illustrates accelerator prompters.

Edit	F3
Cut	Shift+Del
Copy	Ctrl+Ins
Paste	Shift+Ins
Clear	Del

Enabled and Disabled Options

- Whenever using a menu option is not appropriate, the application must disable the option. Display disabled option names in gray type. For example, in the Text application, the user must select text before assigning attributes to it. Therefore, all character attribute options in the Text Menu (Plain, Bold, and Underline) are grayed until text is selected.
- If the user presses F1 while a grayed option is highlighted, the application should display help information about the option.

Classes of Menu Options

When the user selects a menu option, the application might simply perform a task, as indicated by the name of the option, without requiring any further action from the user. This section describes two special classes of menu options, extended options and check options. Using extended or check options can make your application more powerful, more flexible, and easier to use.

Extended Command Options

An option that will request more information from the user is referred to as an *extended command*. When an extended command is chosen, a dialog or message box appears. This box provides the user with information needed for further input. Include an ellipsis (...) after the name of an extended command.

Check Options

A *check option* (also known as a toggle option) has two possible states, on and off. A check mark in front of a menu option indicates that the option is active (selected). The application automatically toggles the state of the option whenever a user selects the option. Check options are often used in groups.

Menu Option Groups

When two or more menu options are related, or provide similar functions, display the options as a group. To visually separate groups within a menu, draw a solid line between groups.

In the Draw application, for example, the Flip horizontal, Flip vertical, and Rotate options on the Actions Menu are related; each of these commands changes the orientation of a selected object. These commands are listed together on the menu and are separated from other commands or groups by a solid line, as illustrated in the following example. Similarly, the Move to top and Move to bottom options appear together, in a separate grouping; these commands move an object to the top or bottom of the print queue. All menu options are left-aligned. Related options are grouped, and groups are separated by solid lines.

Example: The Actions Menu from Draw

Actions	F4
	Duplicate

	Move to top
	Move to bottom

	Flip horizontal
	Flip vertical
	Rotate

	Make object
	Break object

In some cases, the options within a group are mutually exclusive. For instance, the options on Draw's Zoom Menu are mutually exclusive. Only one zoom ratio can be used at any given time. If a user selects an option in such a group, the application must automatically deselect all the other options in the group. A group of mutually exclusive options is analogous to a group of radio buttons (see Chapter 5) in a dialog box.

When a user selects an option in a non-mutually exclusive group, the application must not change the state of any other option in the group. Options in the group must be individually selected to change from on to off, or from off to on. A group like this is analogous to a group of check boxes (see Chapter 5) in a dialog box.

For example, Form Setup's Text Menu allows the Bold and Underline options to be active simultaneously.

Do not mix mutually exclusive and non-mutually exclusive options within a single option group.

Rules and Guidelines for Application Menu Bars

The application menu bar is the interface component that provides access to the options defined in an application. It can include up to nine rectangular buttons. Seven of these buttons are available for custom application menus. The other two are reserved for special DeskMate menus.

Usage and Location

All applications, 80-column and 40-column, must use include a title line and application menu bar. Place the application menu bar immediately beneath the application's title line. The menus on the application menu bar must provide access to all the major functions of the application.

Contents

Each element that can be included in an application menu bar is discussed in the following sections. This section outlines the requirements for application menu bars in 80-column and 40-column applications.

If your application interfaces with the clipboard, its application menu bar must include an Edit Menu. See Chapter 7 for more information about the Edit Menu.

The application menu bar in an 80-column application must contain the following elements:

- Support for on-line help (F1)
- The Message Menu (F9)
- The Accessories Menu (F10)

- Exit and Run options on the F2 menu (usually the File Menu). If the application includes an About option, it must also be included on the F2 menu in its own group.

The application menu bar in a 40-column application must include:

- Support for on-line help (F1)
- an Exit option on the F2 menu (usually the File Menu). If the application includes an About option, it must also be included on the F2 menu in its own group.

The application menu bar in a 40-column application must *not* include:

- The Message Menu (F9)
- The Accessories Menu (F10)

On-Line Help

On-line help is accessed by pressing F1. Do not assign an application menu to F1.

If on-line help is not available about the application, the F1 key accesses DeskMate's on-line help. The Help prompter is displayed on the title line above the application menu bar. It is not a button on the application menu bar or a menu option within a menu.

Exit, Run, and About

All applications must provide Exit and Run commands on the menu accessed by F2. Usually, this menu is called File.

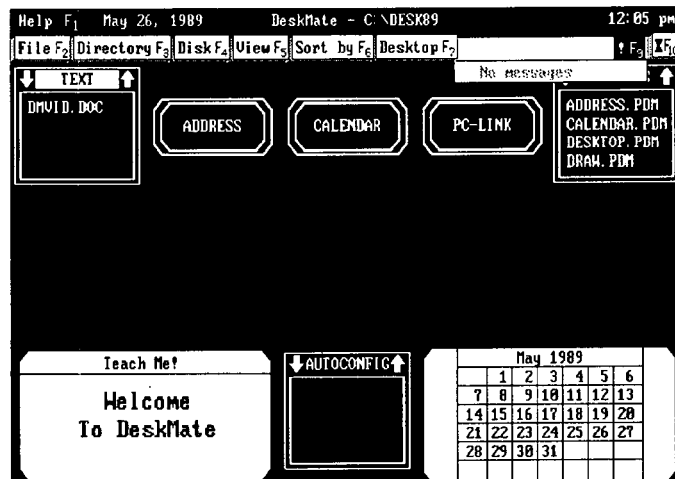
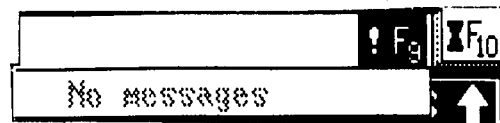
If your application includes an About command, it must also be included on the F2 menu. An About command is recommended but not required. About displays copyright information and other

general information about the application. See Chapter 7 for more information on the File Menu.

The Message Menu

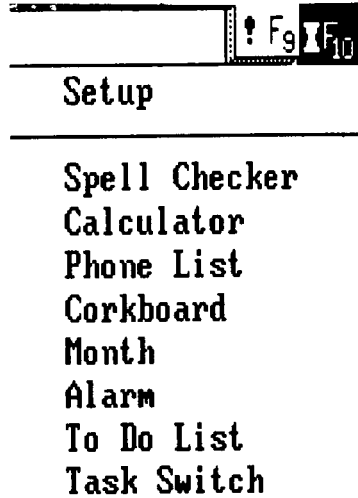
The Message Menu, accessed by F9, displays messages to the user from Calendar, Workgroup, and PC-Link. For example, it notifies the user if an alarm goes off in the Calendar application.

Do not include the Message Menu in 40-column applications.



The Accessories Menu

The Accessories Menu, which is accessed by pressing F10, allows task switching and provides easy access to all currently installed accessories.



Do not include the Accessories Menu in 40-column applications or in menu bar components.

Rules and Guidelines for Accessory Menu Bars

Accessories are not required to use either a title line or a menu bar. If an accessory uses a title line, it must also use a menu bar, and if a menu bar is used, it must meet all the requirements of an application menu bar. However, the converse is not true; an accessory is not required to use a title line simply because it uses the menu bar.

Contents

The accessory menu bar must include:

- Support for on-line help (F1)
- Exit and About commands on the F2 menu (usually the File Menu)

The accessory menu bar must *not* include:

- The Run option
- The Accessories Menu (F10)

Rules and Guidelines for Menu Bar Components

A menu bar component can be used in any pop-up. Menu bar components are not required in any application, but they are the preferred presentation technique in some circumstances. This section describes those circumstances along with the rules and guidelines for using menu bar components.

Rules concerning application menu bars and accessory menu bars are presented in the preceding section.

Usage and Location

A menu bar component can be used in a dialog box. When you use a menu bar component, place it at the top of the dialog box.

Do not use a title line with a menu bar component.

You can include an Exit option in the F2 menu in a menu bar component. This option will provide closure to the dialog box and is an alternative to using the OK and CANCEL buttons in the dialog box.

Contents

Menu bar components must not include:

- the F1 (Help) prompter

NOTE: If the dialog box uses an Exit option in a menu rather than a CANCEL push button, pressing F1 must still access on-line help, even though the prompter is not explicitly shown in the dialog box.

- the Message Menu
- the Accessories Menu

NOTE: F9 and F10 should not be used for any purpose in menu bar components.

Menu Bar Examples

Example: Extended Command Options

The following menu includes extended command options. The ellipsis after an extended command option signals the user that the application requires more information before it can complete the command. Esc, the accelerator for Exit, is printed one space beyond the longest command name, Page Setup.

File	F2
New	
Open...	
Save	
Save as...	
Merge...	
Page setup...	
Print...	
Exit	Esc
Run...	
About ...	

The ellipsis is part of an extended command name. If the longest command is the name of an extended command, accelerators in the menu must be printed one space beyond the ellipsis. If a menu includes more than one accelerator, display the longest accelerator one space beyond the longest command name, and left-align all other accelerators with the longest one.

Example: Check Options

The Text application's Text Menu is an example of the appropriate use of check (toggle) options. On the left, none of the check options is active. On the right, selected options are active (checked).

Text	F5	Text	F5
	Plain		Plain
	<hr/>		<hr/>
	Bold		Bold
	Underline		Underline
	Italic		Italic
	<hr/>		<hr/>
	Center		Center
	Un-Center		Un-Center
	Indent ...		Indent ...

The options in the first two groups describe the weight of printed characters. The first option, Plain, is mutually exclusive with the options in the second group. The options in the second group (Bold, Underline, and Italic) are mutually exclusive with Plain but not with each other. It is possible, for example, to have bold and underline active simultaneously. If any option in the second group is selected, Plain is automatically deselected.

Chapter 5

The Interface Components

This chapter defines each of the components in the DeskMate User Interface. A section called "When to Use" is included for each component. Read this section to determine whether a certain component is appropriate for the task you are trying to implement. After you decide which components to use, read the rules and guidelines for those components.

All DeskMate interface components can be used in pop-up windows and accessories. Some can also be used directly in the work area. The last section of this chapter lists these components, discusses their recommended usage in the work area, and gives an example of a current DeskMate accessory that uses each component in the work area.

General Rules and Guidelines

Whether you use components in pop-ups or in the work area, you can make the screen easier to read by distributing components evenly and minimizing empty space around them. A margin of one character-height or character-width is recommended:

- Between horizontally adjacent components
- Between vertically adjacent components
- On each side of a static box

- Between components within a group
- Between a component group and the box surrounding it

Components and component labels can be drawn in either a normal state or a grayed state. Use the grayed state when the component is disabled. Use the normal state when a component is enabled, whether it is highlighted or not. A component must be disabled whenever selecting it would produce an error condition.

In dialog boxes, components should appear three-dimensional. In the work area, components should appear two-dimensional. Push buttons and icon buttons are exceptions. They must always be three-dimensional so that they can appear in raised (not pressed) or lowered (pressed).

One exception applies to this rule. If the screen design is three-dimensional, the components on the screen should match.

Component Classes

Interface components can be broadly classified as either interactive or static. *Interactive components* convey information to the user, and return information from the user to the application. *Static components* simply convey information, such as a message, to the user.

Interactive components include:

- Check boxes
- List boxes
- Edit fields
- Edit field/list box combinations
- Push buttons

- Radio buttons
- Icon buttons
- Scroll bars

Static components include:

- Text
- Icons
- Boxes

Interactive Components

Interactive components send information to the user and allow the user to send information back to the application. They are used when the application needs information from the user to complete a requested task.

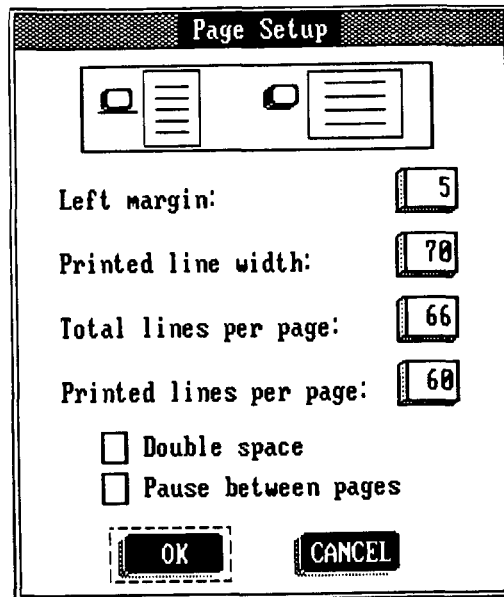
A list box is an example of an interactive component. If the user requests a file operation, such as Open, but does not specify a file name, you can use a list box to allow selection of a file name. The application sends the user a list of existing file names, and the user selects a single file name from the list. After the file name is received, the application continues with the task.

Check Boxes

A check box indicates the state (on or off) of a toggle option. If the toggle option is off, the check box is empty. If the option is on, the box is marked with an X.

When to Use Check Boxes

Use a check box whenever the option can be phrased as a yes-or-no question. Examples are the Double space and Pause between pages options in the Page Setup dialog box.



Use check boxes for short lists (five to six items) only. If you want to present a long list, use a list box (described later) instead.

Check boxes are not recommended for use with lists of file names for two reasons. First, a list of file names is usually too long. Second, the list will probably change every time the application is run. Only static text or static icons can be used with check boxes.

Check boxes are not recommended for use in the work area. Since the application menu bar is accessible from the work area, you

should use checked menu options instead. (See "Check Options" in Chapter 4 for more information.)

Check boxes can also be used to select one or more options from a group of non-mutually exclusive options. Such a group of options is called a *check box group*.

If the options in a group are mutually exclusive, present them as a radio button group, not as a check box group. (See "Radio Buttons," later in this chapter, for more information..)

A user accesses a check box by clicking on the box with the mouse, or by pressing Tab or an arrow key until the check box is highlighted. After highlighting the check box, the user presses the space bar to toggle the state of the check box.

Rules and Guidelines for Check Boxes

A check box must be labeled by text or an icon. The label or icon must clearly identify the purpose of the option. Use an icon when a picture describes the function more clearly and easily than words.

When you use a text label:

- Capitalize the first letter
- Do not use a colon (:) as the last character
- Make the label brief and descriptive

Place a static box around a check box group.

Keep some vertical space between check boxes that are arranged in a column. A space equal to one-half the width of a character is recommended.

List Boxes

A *list box* is used to scan a list of text items, such as file names. DeskMate supports two types of list boxes, single-select and multi-select.

In a single-select list box, the user can select only one item at a time. In a multi-select list, the user can select more than one item. These items may or may not be adjacent in the list.

When to Use List Boxes

Use a list box when:

- A list is too long for radio buttons or check boxes.
- The items in the list are likely to change frequently.

Many DeskMate applications are list-oriented. When a list is too large for the available display space, the application should use a scrolling list box. Scrolling can be performed with the mouse or the keyboard.

List boxes can be used in the work area. Refer to "Using Components in the Work Area," in this chapter, for details.

Rules and Guidelines for List Boxes

The user should be able to select the title of a list box or any item in the list. Selecting the title should produce a different result from selecting a list item. For example, the desktop includes a list box for the Text application. The application name appears at the top of the list box, and the available data files appear in the list box. If the user selects the application name, DeskMate executes Text without a data file. If the user selects a file name, DeskMate executes Text with that file as a data file.

List boxes can be drawn in flat or raised (also called pyramid) style. Flat style is recommended in the work area. Pyramid style is recommended in pop-ups. DeskMate includes functions to draw a list box in either style.

List box items can be displayed in one or two columns. If a single column is used, the cursor should scroll up and down. If two columns are used, the cursor should scroll from side to side.

A list box must be wide enough to display the longest item in the list. The height of a list box should be appropriate to the size of the dialog box and the length of the list. A list box should be tall enough to display at least four items in the list.

In a single-select list box, the user highlights an item by:

- Pressing an arrow key until the desired item is highlighted
- or
- Scrolling through the list with the mouse (by pressing the arrow icons in the title line) until the desired item is highlighted

To select the highlighted item, the user presses Enter or chooses the OK button. The user can also double click on the item to select it and invoke an action once an item is selected.

DeskMate provides a "select and go" feature that allows the user to quickly invoke an action after it is selected. The user has two options to select and go:

- press Enter after an item is highlighted
- double-click on an item

In a multi-select list box, the user can:

- Use an arrow key or the mouse to highlight a single item, as described for the single-select list box

- Press **Ctrl+arrow** to move the cursor to an item without selecting it. Pressing the space bar highlights the item.
- Use **Shift+arrow** (or Shift click with the mouse) to highlight more items.

Edit Fields

An *edit field* is a small text processing window through which the application prompts the user for text input. DeskMate includes edit field definitions for the following types of data:

- Static text
- Right-justified text
- Numbers, with optional decimal places

An edit field can be one line or more than one line. Single-line edit fields can be alphanumeric, numeric, decimal numeric, or expanding alphanumeric. Multi-line edit fields are always alphanumeric. Single-line alphanumeric fields should support side-to-side scrolling. Multi-line alphanumeric fields should support side-to-side and up-and-down scrolling and can provide automatic word-wrapping as well.

When to Use Edit Fields

Use an edit field when you want the user to enter a specific kind of data, such as a date or a time, or a specific amount of data.

Numeric edit fields can be used to verify data format during data entry. Only numeric keys (0 through 9) are accepted during data entry. This can simplify the process for the user and the application.

You can insert static format characters in single-line edit fields to simplify data entry. For instance, if the field will contain a date,

the application can display dashes or slashes between month, day and year values.

Rules and Guidelines for Edit Fields

The size of an edit field should always be appropriate to the area in which it is used. Normally, this means that edit fields used in dialog boxes and accessories are smaller than edit fields used in the work area.

An edit field should not accept values that do not conform to the format of the field. For example, if the user enters an invalid date, the application should reject the entry and prompt the user for a new entry.

When the user enters an edit field, all the data in the field should be selected. To edit data in the field, the user should be able to simply type over existing data; new data should replace the contents of the edit field. When the user moves the cursor within the field, the data is deselected. The user can select a character or a group of characters with the mouse or arrow keys.

The following rules apply specifically to edit fields used in dialog boxes:

- Use only single-line edit fields.
- Use a raised box border around the edit field.
- Label each edit field with a static text string that identifies the text to enter.
- Capitalize the first letter of each word in the label.
- Place a colon (:) at the end of the label. Do not insert a space between the colon and the last character in the label.

When edit fields appear in a column in a dialog box:

- Insert a space one-half the height of a character between fields.
- Place each edit field to the right of its label. Align the left end of all edit fields in a column.
- Align edit fields one character beyond the longest label.

Multi-line edit fields can be used in accessories or in the work area. They should not be used in dialog boxes. Refer to "Using Components in the Work Area," in this chapter, for details on using edit fields in the work area.

Edit Field/List Box Combinations

An *edit field/list box combination* includes an edit field and a list box that work together. As the user scrolls through the contents of the list box items, the currently highlighted item is displayed in the edit field.

When to Use Edit Field/List Box Combinations

Use an edit field/list box combination when the user either can choose an item from a long list or can enter a value not currently in the list.

Rules and Guidelines for Edit Field/List Box Combinations

The list box title should be based on the plural of the edit field label. For example, if the edit field is called File name, use Files as the list box title.

Push Buttons

A push button is a graphic, interactive component that can be placed in one of two states, selected or unselected. It is labeled with text.

When to Use Push Buttons

A push button can be used to:

- Close a dialog box and proceed with a task
- Close a dialog box and cancel a pending task
- Clear the contents of a dialog box
- Invoke a commonly used action, for example, PREV and NEXT buttons while paging through a file.

Push buttons can be used in the work area. Refer to "Using Components in the Work Area," in this chapter, for details.

Rules and Guidelines for Push Buttons

Push buttons can be drawn in a raised or lowered position.

- Draw push buttons in the raised state any time they are unselected.
- Draw a push button in the lowered (selected) position while the application performs the action described by the button. Re-draw the push button in its raised position when the action is complete.
- When a push button is disabled, draw it in the raised position.

The standard accelerator for a push button is Alt+the first letter of the push button label. For example, in Calendar, the accelerator for Previous is Alt+P.

- Use standard push button accelerators unless two push button labels begin with the same letter.
- If you use standard accelerators, do not display the accelerator. If the accelerator is non-standard, display it as part of the push button label.
- If Ctrl is part of a push button accelerator, use the notation `^ <letter>` instead of `Ctrl+letter`. The second notation is used for menu option accelerators only.
- Use only upper case letters in push button accelerators.

Make all push buttons in a dialog box (or a screen in the work area) the same size. Push buttons should be two characters longer than the longest push button label. For example, if CANCEL is the longest label, make all the push buttons in the dialog box large enough to hold an eight-character string.

Use action words or phrases to label push buttons.

- Capitalize all letters of action words, such as RESET.
- Capitalize the first letter of action phrases, such as "Add to sort."
- In the DeskMate Environment, the functions that display push buttons will automatically center the label in the button. It is not necessary to pad the label with spaces.

To push a push button, the user can:

- Use the arrow keys or the TAB key on the keyboard to highlight the button, then press the Enter key

- Move the mouse pointer to that button on the screen, and then click the mouse button
- Use the push button accelerator

Special Push Buttons - OK and CANCEL

OK and CANCEL are usually used together in dialog boxes. OK closes a dialog box and proceeds with the task that called the dialog box. CANCEL closes the dialog box and returns to the application without taking any action.

OK and CANCEL can also be used individually in message boxes, to allow acknowledgment of a message or to halt an operation that would cause an error.

- When OK and CANCEL are used side by side, put the OK button to the left of the CANCEL button, as viewed by the user.
- When OK and CANCEL are used in a column, put the OK button above the CANCEL button.
- Use ENTER as the accelerator for the OK button.
- Use ESCAPE as the accelerator for the CANCEL button.

Radio Buttons

A *radio button* is a component that selects one option from a mutually exclusive group. The group of options is called a *radio button group*.

A radio button works like the select buttons on a car radio. You cannot tune the radio to more than one station at a time; you cannot select more than one option in a radio button group.

When to Use Radio Buttons

Use a radio button group when only one of a small group of options can be active at a time. Do not use radio buttons for large groups of options. Use a single-select list box for a large list.

Rules and Guidelines for Radio Buttons

Do not use accelerators with radio buttons.

To enter a radio button group:

- press the Tab key

To navigate within a radio button group:

- use the arrow keys

- or

- use the mouse

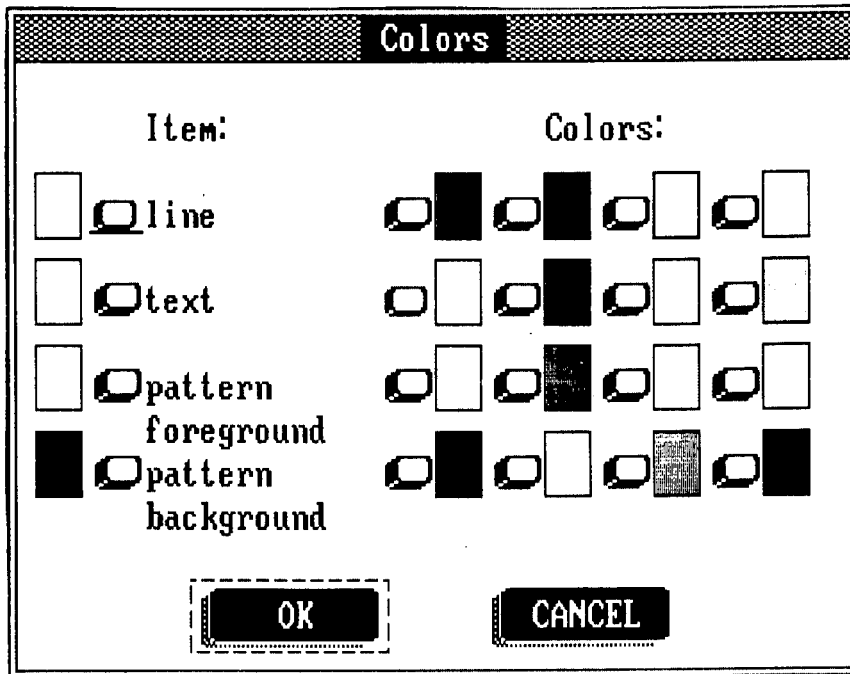
To push a radio button:

- press the Enter key when the desired radio button is highlighted

- or

- click the button with the mouse

Use radio buttons to select options that are represented by icons, such as the Patterns or Colors selections in Draw. The following illustration shows the Colors radio box.



Do not use radio buttons if the options in the group might change. For example, if you want the user to select a file name, do not use radio buttons unless the file names will always be the same. If the file names can change, use a list box instead. Radio buttons are appropriate for selecting a device, such as a communications port. A computer will always have the same ports, but the user might not always want to use the same one.

The group and each radio button in it should have a label.

- Use static text to label the group. A static box around the label is optional.
- Use static text or static icons to label individual radio buttons.
- If static text is used to label individual buttons, capitalize each word in the label.
- Do not use a colon (:) as the last character in the label.
- Commonly recognized acronyms or abbreviations (such as asap, AM/FM, or a.m./p.m.) can be spelled in either upper- or lowercase letters.
- Use static icons whenever text labels are awkward or long. For example, the Patterns and Brush Shapes dialog boxes in Draw use pictures of line styles and brush sizes rather than descriptions of them. In the Text application's Print Menu, the Page Setup dialog box uses pictures to denote portrait and landscape print modes.

Enclose radio button groups in a static box. Do not include any component that is not a member of the group inside the box.

Arrange radio buttons in rows and columns instead of a long list. This way, the user can navigate the group with all four arrow keys, instead of two.

Icon Buttons

An *icon button* functions the same as a push button with a static icon label. A picture drawn on the top of an icon button identifies the purpose of the button.

An icon button is different from a *static icon*. A static icon only identifies a component, while an icon button is an interactive component. For example, the Draw tools are icon buttons.

When to Use Icon Buttons

An icon button is used to select a state or mode for an application. Icon buttons can be used wherever push buttons are appropriate. Use icon buttons whenever a picture describes the button's action more easily or more clearly than words.

Icon buttons can be used in the work area. Refer to "Using Components in the Work Area" in this chapter for details.

Rules and Guidelines for Icon Buttons

Icon buttons can be drawn in raised or lowered position. When an icon button is disabled or unselected, draw it in the raised position. When an icon button is selected, draw it in the lowered position.

Scroll Bars

A *scroll bar* allows quick movement through a list that will not fit on one screen or in one window. Scroll bars are controlled by the mouse, as described in this section. Scrolling can also be performed from the keyboard; see Chapter 2 for details.

A scroll bar consists of:

- Two arrow buttons
- The scrolling region
- The scroll elevator

When to Use Scroll Bars

Use a scroll bar when the information you want to display does not fit on a single screen or window.

Rules and Guidelines for Scroll Bars

Scroll bars can be drawn horizontally or vertically. Horizontal and vertical scroll bars can be used together for very large lists. Place horizontal scroll bars at the bottom of the screen; place vertical scroll bars at the right.

The arrows on a horizontal scroll bar point to the left and right. The arrows on a vertical scroll bar point up and down.

The distance of the elevator from the end of the scrolling region indicates the relative distance of the cursor from the beginning of the list. The beginning of the list is the upper left position in the list. For example, if three-fourths of the scrolling region is above the elevator, it means that three-fourths of the list is above the cursor.

A scroll bar responds to the mouse as follows:

Clicking on an arrow button

Moves the cursor through the list one item at a time in the direction indicated by the arrow. This is equivalent to pressing one of the arrow keys on the keyboard.

Clicking in the scrolling region

Moves the cursor through the list one page at a time in the direction indicated by the arrow. This is equivalent to pressing Page Up (up one page), Page Down (down one page), Ctrl+Page up (left one page), or Ctrl+Page down (right one page) on the keyboard.

Dragging the elevator

Moves the cursor quickly through large sections of a list. This is equivalent to pressing the Page Up or Page Down key. Dragging the elevator to the top of the scrolling region has the same effect as pressing Ctrl+Home. Dragging the elevator to the bottom of the scrolling region has the same effect as pressing Ctrl+End.

Static Components

Static components, unlike interactive components, simply send information to the user. They do not return information to the application. The application will not take any action when static components are selected. Use static components to make the user aware of special circumstances, such as an error or an invalid file type.

Text

Static text is a word, or a group of words, that labels an interactive component or sends a message to the user. Printer is out of paper is an example of a message. Although such a message calls for action by the user, the user is not required to furnish information to the application before the task can be completed.

When to Use Text

Use text to:

- Create labels for radio button groups, check boxes, and edit fields
- Give instructions for using an application, an accessory, or a dialog box
- Alert the user to error conditions

Rules and Guidelines for Text

Text can be drawn in two states, normal and grayed. When text is used as a component label, draw the text in grayed state if the component is disabled. Draw the text in normal state if the component is enabled.

Leave one space between a text label and the component.

Icons

A static icon is a picture that describes an option or setting. Icons can be created by including graphics lists in the application code, by creating a bitmap, or by creating any other graphic output form that can be included in the program.

A static icon is different from an icon button. A static icon is simply a label; selecting an icon button will invoke an action.

When to Use Icons

Use icons whenever a text label would be awkward or unreasonably long. Icons are also useful when space is limited.

Rules and Guidelines for Icons

Use icons with radio buttons and check boxes.

Boxes

A static box is used to surround a group of related components. Static boxes help the user quickly identify component groups on the screen.

When to Use Boxes

Use boxes to surround radio button groups, check box groups, or edit fields.

Rules and Guidelines for Boxes

Boxes can be drawn in flat or raised style.

Using Components in the Work Area

Most interface components are recommended only for use in dialog boxes and accessories. List boxes, edit fields, push buttons and icon buttons are the only components that are recommended for use in the work area. This section describes the rules and guidelines for using these components in the work area.

When interface components are used directly in the work area, the application menu bar is not disabled. This means the user can access menu options while components are active on the screen. This is different from dialog boxes. While dialog boxes are open, the menu bar and the rest of the screen are disabled. The user must close the dialog box before using any menu option.

Static components can be freely used in the work area.

List Boxes

Use list boxes for long lists and lists that change frequently.

Edit Fields

Use edit fields in the work area for text entry and editing.

The only real requirement in the work area is that the screen must be well organized and easy to use. Edit fields used in the work area should be labeled.

In accessories, edit fields must be clearly identified, but not necessarily with a static text label. For instance, the edit fields used in Address Book look like an address book entry. The user can easily identify the expected entry from the graphic context of the screen. A text label is not necessary.

Push Buttons

Push buttons follow the same appearance and operations rules in the work area as in dialog boxes.

Icon Buttons

Use icon buttons in the work area to change the state of the application or accessory. For example, Draw uses icon buttons to change the drawing mode or the drawing style.

Chapter 6

Pop-Ups

This chapter defines A DeskMate pop-up, or pop-up window. It describes the types of pop-ups supported, and describes the rules and guidelines that apply to pop-ups.

A pop-up, or pop-up window, is a group of standard DeskMate user interface components. A pop-up can be used to get information to the user, send a message to the user, or enable the user to perform a specialized task. A pop-up appears under specific circumstances, accomplishes a specific goal, and then returns control to the application. Any interface component described in Chapter 5 can be used in a pop-up.

As long as a pop-up is active (displayed on the screen), the application menu bar and the application work area are disabled. The user must close the pop-up before the application will be enabled again.

When to Use a Pop-Up

Pop-ups are useful in a wide variety of circumstances, provided your task meets a few basic criteria. If the task does not meet these criteria, try another type of interface. It might be possible to break your task into smaller subtasks, each of which can be processed in a single pop-up window.

Use a pop-up if:

- A specific task needs to be performed during execution of an application.
- The task does not require use of the application menu bar or any of the work area outside the pop-up window. Both the application menu bar and the rest of the work area are disabled as long as the window is open.
- All the information necessary to complete the task is available inside the pop-up window.
- Closing the pop-up window returns the user to the application. If the pop-up is interactive, the state of the application might be different after the pop-up is closed.
- Closing the pop-up window does not terminate the application. When a pop-up is closed, the application resumes normal processing. A pop-up that returns to the application but does not terminate the application is said to have *closure*. Closure is a required property of all pop-ups.

Types of Pop-Ups

Pop-ups fall into three broad categories: *message boxes*, *dialog boxes*, and *accessories*.

A message box is a *non-interactive* pop-up. It is used simply to convey information to the user. The user acknowledges receiving the information, but can take no further action before closing the window. Message boxes are appropriate for cautions and error messages (such as File not found) and status reports during lengthy operations (such as Printing in progress).

A dialog box is an *interactive* pop-up. It enables the user to make decisions and converse with the application before closing the window. (*Dialogue*, or conversation, occurs between a user and an application in a *dialog* box.) Depending on the decisions made by

the user, a dialog box might alter the state of the application when it closes.

An accessory is an interactive pop-up that enables the user to perform a specialized task. Accessories are the most elaborate type of pop-up. Calculator and Phone List are examples of pop-up accessories.

Message Boxes

Message boxes are the simplest pop-ups. Their only interface components are static text and push buttons. Usually, they contain only one button, the OK button, which is used to acknowledge the message.

The following illustration shows a message box. Notice the OK button centered at the bottom of the box.

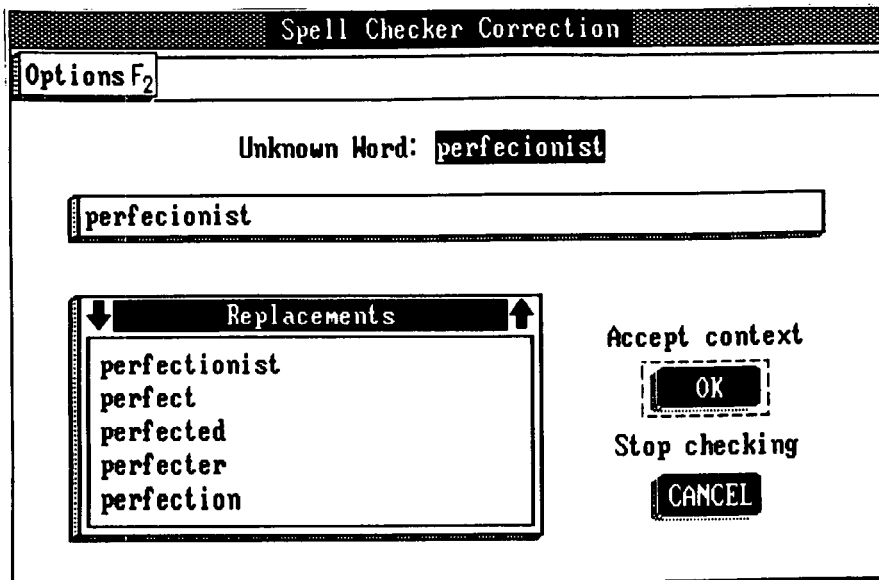


DeskMate applications must save screen background information before displaying a pop-up and restore screen information after a pop-up is closed.

Dialog Boxes

Dialog boxes focus the user's attention on a specific task and enable communication between the user and the application. All information necessary to complete the task is presented in the dialog box.

The following illustration shows one of the dialog boxes in the Text application. This box is from the Spell Checker accessory used in Text.



Dialog boxes are appropriate for a variety of tasks, such as proofreading a document for spelling errors. A dialog box can focus the user on spell-checking the document. While spell-checking, the user cannot do anything but correct spelling errors. When the user completes the task, all words are spelled correctly, and the application returns to normal operation.

Accessories

Accessories often look like dialog boxes, but they are more like miniature applications. They are not bound by the same appearance and operation rules as dialog boxes. Use accessories to complete concise, self-contained tasks. For example, a spell checker in a word processing program is suitable for an accessory. A spell checker can also be implemented as an application dialog box. The advantage of an accessory spell checker is that you can use it in any application that has access to the Accessories Menu. If the spell checker is a dialog box within an application, it is not accessible while that application is not running.

Rules and Guidelines for Pop-Ups

Any DeskMate interface component can be included in a dialog box or an accessory. The components that can be included in message boxes are more limited. This section describes the rules that apply to size, position, and usage of components within the various types of pop-ups.

Size and Position

Always try to position pop-up windows in the same location on the screen. They also must fit within the application work area. The preferred location is the middle of the work area, centered horizontally and vertically. A pop-up should not occupy the entire work area unless absolutely necessary.

Because they are focused tasks, all pop-ups should be visually bounded by frames. The frame of a pop-up overlays the screen of the application. Information beneath the pop-up should remain visible to the greatest extent possible.

Size and Position of Dialog Boxes

Always try to display dialog boxes in the center of the screen. Using a single location for dialog boxes gives your application a consistent, predictable appearance. Display a dialog box in a different location only if you have no alternative, for example, if centering it will obstruct the user's view of information needed to complete the box.

Keep pop-ups (dialog boxes in particular) as small and simple as possible. Since pop-ups are intended for very specific tasks, small, simple presentation helps to focus the user's attention on the task at hand. Large, complicated boxes are difficult to understand. Try to group information from very large boxes into two or more small boxes. For example, do not group printer information and page layout information together in a single dialog box. Make two boxes, one for printer information and one for page layout information.

Size and Position of Message Boxes

Message boxes must be centered, horizontally and vertically, on the screen. If you use DeskMate functions, the size and position of a message box is controlled by the system.

Titles

- Use the title to describe the function of the pop-up as precisely as possible.
- Capitalize the first letter of each word in the title.

- Do not punctuate the title.
- Do not provide instructions in the title.

Dialog Box Titles

Every dialog box must have a title, which appears in the dialog box frame, that identifies the action or procedure the dialog box performs. For example, the Open option in the File Menu displays a dialog box titled Open File, and the Page setup option displays the Page Setup dialog box.

If a dialog box requests additional information for an extended command, repeat the menu option name in the dialog box title. Extended commands are described in Chapter 4.

Message Box Titles

Every message box should also have an appropriate title.

If the message box was invoked by a menu option, use the option name in the title of the message box.

Message Text

The message should describe the problem or the action taken as clearly as possible.

- The text must be no longer than three 30-character lines. Messages are word-wrapped automatically.
- Use complete sentences, including appropriate punctuation.
- If a message notifies the user that a task, such as printing, is taking place, the message should end with an ellipsis (...).

For example, Printing in progress..., with a CANCEL button tells

the user that printing is being done and the user can cancel the task at any time while the message is displayed.

- Do not use contractions. For example, use *do not* and *cannot* instead of *don't* and *can't*.
- Avoid punctuation such as colons and exclamation points.
- Do not include the application name in the message.
- Use all capital letters on file names in the message.
- Capitalize the first letter of any key names in the message.
- Use accelerators sparingly to avoid complicating the dialog box interface. Refer to Chapter 5 for more information about using and defining accelerators.

Pop-Up Operation

This section describes the general behavior that is expected from pop-ups. It includes error handling procedures, and some information about displaying and removing pop-ups during an application.

Default States

Always provide a default state or value for each check box, list box, radio button, and edit field in a dialog box.

- Dialog boxes should "remember" previous settings whenever possible.

Disabled Components

If a component is disabled, draw the component and its label in the unselected state. Disable components when selecting the

component is an invalid operation for the application. The unselected state is:

- Off, for a check box
- Empty, for an edit field
- Raised, for a push button or radio button

Exception: If all buttons in a group are disabled, the default button must be pressed.

Component Behavior

A list box is the only component in a dialog box that enables the user to select an item and immediately terminate the dialog box.

When the user presses the Enter key or double clicks on a list box item, the application should choose the highlighted item and select the OK button.

Draw a push button in its raised position initially. When the user selects the button, draw the button in its lowered position. Once the application completes the process invoked by the button, redraw the button in its raised position.

When an edit field/list box combination is used, display the currently highlighted list item in the edit field.

Push Buttons in Message Boxes

The following push button combinations are supported in message boxes:

- OK
Enables the user to acknowledge a message.

CANCEL

Enables the user to stop an operation.

OK, CANCEL

Enables the user to continue or cancel the operation.

YES, NO

Enables the user to answer a prompt with yes or no. The prompt must be phrased as a question and end with a question mark.

YES, NO, CANCEL

Enables the user to cancel the operation, or answer yes or no and continue. The prompt must be phrased as a question and must end with a question mark.

RETRY, CANCEL

Enables the user to cancel the operation or try it again.

Assigning Default States

Valid default settings should always be given to the components within a dialog box, enabling the user to accept the information (OK button) without causing errors to occur.

When the user selects (pushes) the OK button, the application should verify the accuracy and format of data accepted from the user. When the user enters invalid data, such as an invalid file name in an edit field, the application should:

1. Display a message box informing the user of the error and giving the user the option to retry or cancel.
2. Highlight the invalid data or the component that contains the invalid data.
3. Raise the pushed button.
4. Rerun the dialog box, enabling the user to try again.

Removing Dialog Boxes

When memory permits, the application should save the screen background and restore it later to remove the dialog box. If there is insufficient memory, the application should redraw the screen to remove a dialog box.

Pop-ups can be *stacked*, or *layered*, as long as they are unstacked in reverse order. If the closure of one pop-up returns to a previous pop-up, the second pop-up should completely cover the first pop-up. This way, users will not be confused into thinking that some components in the first pop-up are available while the second pop-up is active.

User Interfaces to Pop-Up Windows

The user must be able to access pop-ups with the keyboard as well as with the mouse. This section describes the required elements of each interface.

Keyboard Interface

Esc (Escape)

Selects the Cancel button to terminate the pop-up with no action taken.

Enter

If the highlighted component is not a push button, pressing Enter presses the OK button to terminate the pop-up with affirmative action. If the highlighted component is a push button, pressing Enter pushes the button.

If the highlighted component is a list box, pressing Enter selects the component and immediately executes the task.

Space bar

If the highlighted component is a check box, pressing the space bar toggles the state of the check box.

If the highlighted component is a push button, icon button, or radio button, pressing the space bar selects the button.

Tab

Moves the highlight forward to the next component or component group.

Shift+Tab

Moves the highlight backward to the next component or component group.

Up Arrow

Highlights the nearest component above the current component (if one exists). After the user presses the arrow keys to leave the component, pressing the up arrow highlights the previous element of the component. The state of a component does not change when the user arrows out.

Down Arrow

Highlights the nearest component below the current component (if one exists). After the user presses the arrow keys to leave the component, pressing the down arrow highlights the previous element of the component. The state of a component does not change when the user arrows out.

Right Arrow

Highlights the nearest component to the right of the current component (if one exists). After the user presses the arrow keys to leave the component, pressing the right arrow highlights the previous element of the component. The state of a component does not change when the user arrows out.

Left Arrow

Highlights the nearest component to the left of the current component (if one exists). After the user presses the arrow keys to leave the component, pressing the left arrow highlights the previous element of the component. The state of a component does not change when the user arrows out.

Mouse Interface

The effect of clicking the mouse button on a component depends on the type of component:

Click

Clicking on a check box toggles the state of the box.

Clicking on a radio button, an icon button, or a push button selects the button.

Clicking on an edit field activates the edit field as if the user pressed the Tab key or an arrow key to enter the field.

Double Click

Selects and immediately executes a list box item.

Drag

Dragging the mouse selects a group of characters in an edit field.

Chapter 7

Special Menus

This chapter describes some particular functions you can include in your application and the DeskMate menus that you must include to support those functions.

Message

DeskMate uses the Message to relay messages from the system to the user. Include the Message Menu as part of the application menu bar of all 80-column applications. Do not include the Message Menu in 40-column application menu bars. The Message Menu is optional in menu bar components used in dialog boxes, and in accessory menu bars.

Accessories

The Accessories Menu is a list of all DeskMate accessories currently loaded on the system. Include the Accessories Menu in 80-column application menu bars. Do not include it in accessory menu bars, 40-column application menu bars, or menu bar components.

File

The F2 menu, usually called File, provides access to file management functions. All 80-column applications must include Exit and Run options on the F2 menu. If your application includes an About option, you should also include it on the F2 menu. Some examples of other options you may want to include are:

- New, Save, Save as, and Open, if the application involves creating and using new files
- Print, if the user can print files from the application
- Page setup, if printing will require special formatting
- Merge, if the user might combine multiple files for a single operation

Other special file functions in your application might suggest other options that can be included in the File Menu.

A typical File Menu follows:

```
File F2
New
Open...
Save
Save as...
Merge...
Page setup...
Print...
Exit          Esc
Run...
-----
About ...
```

The rest of this section describes the operation of each option included in the preceding example.

New

The New option must be enabled whenever the user can create a new file. If a data file is already in use, it prompts the user (with a YES/NO/CANCEL message) to save any unsaved, modified data.

- If the user selects Yes, the data is saved to the current file. If the file is untitled, the Save as dialog box will appear. See "Save as" option description which follows for more information. After the file is saved, the application clears working memory and gets ready for a new file.
- If the user selects No, the current data is not saved, the application clears memory and gets ready for new data.
- If the user selects Cancel, the New option is terminated and the application retains the current data file.

The New option sets the application to its default state, or the Untitled state, and "Untitled" is displayed as the file name.

If the application does not allow the Untitled state (such as in some database applications), the application should prompt the user for a valid file name. In this case, the New option appears with an ellipsis (...) because the user must supply additional information (the file name).

Open...

The Open option must be enabled whenever a file can be loaded. If a file is already in use, the application prompts the user (with a YES/NO/CANCEL message) to save any unsaved, modified data.

The option's responses to YES, NO, and CANCEL are similar to the responses of the New option. The option will proceed if the user selects YES or NO, and will terminate if the user selects CANCEL.

After the message is answered, the application displays an Open File dialog box and prompts the user for a file name. The user

must select OK or CANCEL in the Open dialog box. If the user selects OK, the file is opened and loaded into memory. If the user selects CANCEL, the application returns to its prior state.

Applications that automatically update files as data is entered do not need to prompt the user to save changes before displaying the Open File dialog box.

Save

Save must be enabled whenever the current file can be saved without first prompting the user for more information. It is disabled when the application is in an Untitled state.

Do not include this option if data updates will occur automatically without a specific user request. Include it if the user will be allowed to make periodic updates to data on the disk.

Save as...

The Save as option must be enabled whenever the user can change the data in an existing file and then save the changed data in a different file. The application displays a Save File dialog box and prompts the user for a file name.

If the user selects OK in the Save File dialog box, the application should verify the file name. If the specified file already exists, the application should display a yes/no/cancel message box, to ask whether the user wants to overwrite the existing file.

If the user selects CANCEL in the Save File dialog box, the application should return to its prior state.

Merge...

The Merge option enables the user to combine the information in one file with the information in another file. It displays a Merge File dialog box and prompts the user for the file to merge in.

If the application enables the user to merge data from another source into the current file, the application should display a dialog box that prompts the user for information about the source.

Merge operations should move specified data into the current file. Do not move the current file into some other file.

Page setup...

The Page setup option enables the user to define the printed page layout for the current file. The application displays the Page Setup dialog box, prompting the user for information such as margins, total lines per page, printed lines per page, and printed line width.

Print...

The Print option enables the user to print the current screen or file to the printer, the screen, or a file. The application uses the Print File dialog box to get print information from the user.

If you want to customize printing options, you can use a phrase to identify this option, such as Print Page, or Print Drawing. In this case, the phrase identifies clearly what will be printed. It is not required that a printing option support screen, printer and file as output destinations. You can choose to support some of them and omit others, as appropriate to your application. If you want to use a name other than Print, always begin the name with "Print."

The accelerator for this option is the Print key. It is not noted in the menu since it is a machine-specific accelerator.

Exit

The Exit option is required for all applications. This option closes the current application and returns to the desktop. If the application was started from a Runtime rather than from the desktop, Exit returns to DOS. The Esc key is the accelerator for Exit.

Before closing the application, Exit checks to see whether the data in memory has been modified since the last save operation. If it has been modified, Exit prompts the user to save the changes. If the user cancels the operation, the application returns to its previous state.

Run...

The Run option enables the user to run a different application without first returning to the desktop or DOS. This option displays the Run dialog box. After the user fills in the box and selects OK, Run exits the current application and starts the application specified in the Run dialog. If the user selects CANCEL, Run returns to the current application.

If a data file is open under the current application and that file contains unsaved changes, the Run option prompts the user to save the file before proceeding.

About...

The About option displays a dialog box that contains the application's name, version number, and copyright information. Capitalize the first letter of each word in the application name. This option is recommended for all applications and accessories.

The About option should be placed in a separate menu option group.

Edit

Applications that use the DeskMate clipboard must have an Edit Menu. The Edit Menu enables the application to transfer data to and from the DeskMate clipboard.

Edit	F3	
Cut		Shift+Del
Copy		Ctrl+Ins
Paste		Shift+Ins
Clear		Del

Your application should include an Edit Menu if the user can move or copy data to another area of the same file or to a different file. For instance, figures from a spreadsheet could be transferred to a word processor for a business forecast report.

Applications that enable the user to move and copy selected data between files or to other applications must use the DeskMate clipboard instead of a data buffer. Information in a data buffer cannot be transferred to other files or other applications.

When the user starts an application or performs a task switch, the application should check the clipboard contents. If the clipboard contains data that is valid in the application, the application should enable the Paste option.

DeskMate applications that interface with the clipboard should provide a way for the user to "select" the information. Highlight selected text in reverse video. Outline graphics with a handle box.

The Edit Menu should contain the following options, in the following order:

Cut

The Cut option is enabled only when data is highlighted. It removes the selected data from the file and places it on the clipboard. Shift+Del is the accelerator for Cut.

Selecting the Cut option disables Cut, Copy, and Clear, and enables Paste.

Copy

The Copy option must be enabled only when data is highlighted. It copies the selected text to the clipboard without removing it from the current location, and deselects the data. The user must select the text again if the text is required for another operation. Ctrl+Ins is the accelerator for Copy.

Selecting the Copy option disables Cut, Copy, and Clear and enables Paste and Select all.

Paste

The Paste option must be enabled whenever the clipboard contains data that is valid in the current application. Paste copies the clipboard contents to the current cursor position or to a position specified by the current application, and deselects the data in its new location. Shift+Ins is the accelerator for Paste.

For example:

The CLIP_DRAW data type is not supported by the Worksheet application, so the Paste option in Worksheet is grayed when CLIP_DRAW data is on the clipboard.

The CLIP_DRAW type is supported by the Form Setup and Text applications, so the Paste option in either application is enabled when CLIP_DRAW data is on the clipboard.

If the data can appear more than once in the file, the option remains enabled after the Paste operation is complete. If the data can appear only once in the file, the option is grayed and the contents of the clipboard are cleared by the application.

Clear

The Clear option must be enabled when data is selected. It removes the selected data from the screen and the file without affecting the contents of the clipboard. Clear is accelerated with Del.

Selecting Clear disables Cut, Copy, and Clear and enables Paste (if the data is valid in the current application) and Select all.

**DeskMate Development System
Development Guide
03.05.00**

About the Guide

This guide covers all aspects of DeskMate development, from choosing a development system and memory model, through the implementation of the application by using the examples and tools, and finally the distribution of your application and providing help, tutorials, and demos for your product.

Getting Started contains the preliminary information needed before development begins. Important decisions made early on can effect your development schedule and the success of your product when it is complete. This section introduces several concepts important to DeskMate development and covers compatibility and programming issues which you should be aware of. You should read this section before development begins and refer to it during the development process.

Programming Examples, describes each of the samples provided in the kit and covers some special programming topics of interest. We recommend using one of the samples as a template or starting point when developing a DeskMate application.

Tools and Utilities provides user documentation on how to actually use the tools provided with the development system. These tools help reduce the time it takes to create a working application. You can create menus, dialog boxes, bitmaps, custom fonts, and pictures with these utilities. You can also analyze the memory requirements of your application.

Distributing Your Application covers the DeskMate Checklist and how to write your installation programs. It also provides guidelines you should use when determining what your runtime diskette file distributions should be.

DeskMate Help Systems describes the help available in a DeskMate environment and how your application provides help to the user. The new Intelligent Help Manager which provides context-sensitive help is described in great detail. Writing Tutorials and Demos describes how to use the DeskMate Tutorial Technology to author tutorial scripts and demos for distribution with your product. The documentation for the tools and utilities provided for each of these systems is included in the discussion.

Contents

About the Guide

Part 1 - Getting Started

Introduction	1-1
Memory Models and Development Tools	1-5
DeskMate Coordinate Systems	1-13
Compatibility and Programming Issues	1-23
Overview of the Tools, Utilities, and Examples.....	1-29

Part 2 - Programming Examples

A DeskMate Shell	2-1
Using the DeskMate Coordinate Systems	2-7
DeskMate File Handling	2-13
Printing	2-47
Using the Graphics Form Manager	2-57
Special Topics	2-65
Writing a 40 Column Application	2-77
Writing a DeskMate Resource	2-79
Writing a DeskMate Accessory	2-89

Part 3 - Tools and Utilities

Menu bar Builder	3-1
Dialog Box Builder	3-3
Bitmap Editor	3-7
Graphics Form Generator	3-9
Clipart File Builder	3-11
Stroke Font Editor	3-13
Memory Map Generator	3-15
Desk Header	3-17
Disk Label Generator	3-21
Customized Runtime Utility	3-25
Customized INSTALL.EXE Utility	3-26

Part 4 - Distributing Your Application

The DeskMate Checklist	4-1
Installation and Upgrade Procedures	4-3
Determining DeskMate Product Versions	4-5
Runtime Distribution Guidelines	4-7

Part 5 - DeskMate Help Systems

Overview	5-1
Writing the Application Help File	5-5
Creating the Sample Help File VIDEO.HLP	5-13
Help Rule Base Utility	5-25
DeskMate Help Editor	5-29
Help File Compression Utility	5-31
Help File Format	5-33

Part 6 - Writing Tutorials and Demos

The DeskMate Tutorial Technology	6-1
Authoring a Tutorial Script	6-3
The DeskMate Introductory Tutorial	6-5
Script Command Reference	6-41
Tutorial Player	6-105
Demo Launcher	6-106
Event Recorder	6-107
Script File Interpreter and Compiler	6-108
Tutorial Compression Tools	6-109

Appendix A - DeskMate 3 Application File Formats

Introduction	A-1
Address Book/Phone List	A-3
Calendar	A-5
Draw	A-9
Filer/Form Setup	A-11
Text	A-15
Worksheet	A-21

Part 1
Getting Started

"Getting Started"

Contents

Introduction	1-1
Memory Models and Development Tools	1-5
Memory Models	1-5
Development Tools	1-5
Compiling	1-5
Linking	1-6
Debugging under DeskMate	1-6
Using Turbo Debugger	1-6
Using SYMDEB	1-8
Using CodeView	1-9
Using Periscope	1-11
DeskMate Coordinate Systems	1-13
About World Coordinates	1-13
Using World Coordinates	1-14
Using World Extents	1-14
World Coordinates - General	1-15
Normalized World Coordinates	1-15
Point to Point vs. Origin Extent	1-16
Origin Independent Extents	1-16
Finding Adjacent Pixels	1-17
Finding the Nth Pixel	1-17
World, Viewport, Clip	1-19
World and Viewport Relationship	1-21
Clip Regions	1-21
Window Manager	1-21
Compatibility and Programming Issues	1-23
Runtime Executive	1-23
The Help System	1-24
DeskMate 3.0 Operation	1-24
DeskMate 3.3 Operation	1-24
Compatibility Issues	1-24
The F10 Tandy Menu	1-24
Code Shedding when Running Accessories	1-25

DeskMate 3.0 Code Shed Operation	1-25
DeskMate 3.3 Code Shed Operation	1-25
Programming and Compatibility Issues	1-25
"Sticky Menus" and Selectable Grayed Menu Items	1-26
Animated Busy Icon	1-26
Form Manager and GUF Resource	1-27
Loading of the Resources for 3.0 Applications	1-27
Loading of the Resources for 3.3 Applications	1-27
Video Drivers	1-27
Driver Names	1-27
Video Detection	1-27
Palettes	1-27
Printer Drivers	1-28
Line Styles	1-28
Print regions	1-28
Landscape printing	1-28
Overview of the Tools, Utilities, and Examples.....	1-29

Introduction

After reading About This Kit, reviewing the DeskMate Style Guide, and installing your DeskMate 3 product and development system you are ready to develop a DeskMate application. Before you begin development, we should review the key information discussed so far and introduce some new topics which you should find beneficial in the development of your application.

The Kit contains the 1) development files, 2) samples, and 3) tools and utilities need to develop a DeskMate applications. The DeskMate Technical Reference defines every function call available in the DeskMate libraries.

DeskMate applications are primarily written in C but may also be written in assembly language. Programs may be written in any of the memory models but only the small and medium memory models have DeskMate libraries. Refer to Memory Models and Development Tools, in this section, for a detailed discussion of memory models, and compiling, linking, and debugging of DeskMate applications.

The DeskMate Style Guide defines the DeskMate User Interface. DeskMate applications use menus, dialog boxes, message boxes, and interface components to communicate with the user. DeskMate applications support both a keyboard and mouse interface. Your application should meet the DeskMate standards defined in this guide.

From the System Overview in About This Kit, you learned about DESK, the DeskMate Executive, and the key DeskMate resource - Core Services Resource (Core or CSR), and the other resources available in the DeskMate environment. Applications communicate with these resources through the DeskMate libraries.

There are now two versions of DeskMate 3 in distribution, DeskMate 3.0 (includes 3.2) and DeskMate 3.3. Your application should check the system version number, when it is initially loaded, by calling **dm_inquire_product** to determine which version of the environment the application is running on.

Now, let's introduce some new DeskMate programming topics.

DeskMate uses a *world coordinate system* to access the video. In the programming examples and the function call descriptions in the DeskMate Technical Reference you will often see the defines, **CHAR_XEXT** and **CHAR_YEXT** used. These defines allow the programmer to reference points on the screen as character locations. DeskMate also allows the video to be accessed at a pixel or device level. See DeskMate Coordinate Systems for a detailed discussion about world and device coordinates.

DeskMate applications are *event-driven*, they wait for the user to perform an action and then act upon the action. The CSR provides an Event Interpreter or Manager which translates the user's actions into events the application can process. Applications can write their own event interpreters to capture events before and after the CSR's Event Manager has handled them. For more information, see the Event Manager section of the DeskMate Technical Reference.

DeskMate allows mini-applications, called accessories, to pop-up over the current application. When there is not enough available memory to load the accessory, Desk will try to make room for the accessory by getting rid of part of the application's code and moving the rest. This process is referred to as *code shedding*. The following criteria is used to determine if your application can be code shed to run an accessory. If your application cannot be code shed then it MUST call **dm_exec_dont_shed** when initially loaded to insure that it is not code shed to run an accessory. Your application should also set the code shed size using the DeskMate utility DESKHDR.EXE.

- 1) An overlaid application cannot be code shed since it cannot be guaranteed that it will be restored from the disk in the same configuration it was in before the accessory was run.
- 2) An Application which uses event interpreters or interrupt handlers cannot be code shed because the interpreters and handlers are address dependent. When the application is moved during the code shed, the handlers are moved and may no longer function correctly.

Note: On a DeskMate 3.3 system the application may be able to code shed if the handlers are placed in the IMPURE segment which is not altered during a code shed. Refer to the detailed information for DESKHDR.EXE in the Tools and Utilities section of this guide for more information about splitting applications.

- 3) A medium or large model application which has too many fix-ups (more than 200), cannot code shed in a DeskMate 3.0 system but can on a 3.3 systems which supports unlimited fix-ups.

Note: This deficiency in the 3.0 system can be overcome by naming the code segments and limiting the number of code segments used to a smaller number. Refer to your compiler documentation for more information about overriding the default code segment name.

The executive and the resources often use the application's stack. The CSR and its drivers require the application stack for busy icon and mouse processing. A *packed executable* has a very small temporary stack while it is being loaded before the stack is expanded. This stack can be overflowed during the loading of the application if the busy icon or mouse processing consume more of the stack than is available. You should not pack your DeskMate executable and should allow at least 2048 bytes of stack space for the executive and DeskMate resources, and 4096 bytes if the Form Manager Resource is used.

After reviewing the Memory Models and Development Tools section, review the DeskMate Coordinate Systems if your application will be accessing the video to do graphics or if you want to access the video at the pixel level. You will want to review these sections again once you actually start development and are more familiar with the system.

If you have developed or are developing an application using the DeskMate 3.2 Development System, you should read the section on Compatibility and Programming Issues for important information which could affect your application. New developers should also review this section since it introduces many topics which might affect your application.

The next step is to review the Overview of the Tools, Utilities, and Examples to get a good picture of the overall development system. Start with one of the examples, the one that matches your application the best, and expand on it using the tools and utilities supplied with the kit. You are now ready to begin your DeskMate development.

Memory Models and Development Tools

Memory Models

Small and medium memory models are supported through the DeskMate libraries, `DM.LIB` and `DMMED.LIB`. Applications are limited to 64K of data space. The data segment and the stack segment must be the same (`DS == SS`).

Large model DeskMate applications are not directly supported through a library and require additional coding by the programmer. A large model application uses the medium model library, `DMMED.LIB`, to communicate with the executive and resources. The application's DeskMate data and stack must be in the default data segment, `DGROUP`, when the application makes a DeskMate function call.

Certain calls store the address of the application's data for use by other function calls. For instance, the `mb_draw` call saves the menu bar address for use with by the `event_*` function calls. For this reason, the menu bar should not be moved to a different memory location between calls to `mb_draw` and `event_*`. If the menu bar is moved, it must be restored at exactly the same location for the program to function correctly.

Whenever possible, all DeskMate data, the menu bar, dialog boxes, messages, etc., should be defined in the default data segment and the other application's data should be defined in alternate data segments to insure the data used by a DeskMate functions is in the correct data segment.

Development Tools

The Kit does not contain the development tools necessary to write software, an editor, compiler, assembler, linker, or debugger. It only contains those required to write a DeskMate application. We recommend you use one of the following development systems for DeskMate development.

Compilers/Assemblers/Linkers

- Microsoft C 4.0, 5.0, or 5.1 with Microsoft MASM 5.0
- Microsoft Quick C
- Turbo C and Assembler 2.0

Debuggers

- Microsoft's SYMDEB from MASM 4.0
- Microsoft's CodeView
- Periscope
- Turbo Debugger

Compiling

The system resources assume data structures used by their functions are packed or byte aligned. Make sure that you use the pack structures option, `/Zp`, for Microsoft C when compiling your DeskMate source modules. The default data alignment is byte for Turbo C code generation.

The Turbo C 2.00 startup code must be changed because Turbo sets video to 80x25 text mode when that video mode is available. Turbo C makes direct calls to BIOS to set the mode, so DeskMate is unaware of the change. DeskMate 3.3

assumes that a DeskMate application will not change the graphics mode without using a DeskMate video call.

To use the Turbo tools in creating a DeskMate application, the Turbo C startup code (in C0.OBJ) must be reassembled with the symbol `__OLDCONIO__` defined before using the compiler with DeskMate. The routines that are defined with this symbol leave the video mode intact.

For small-model startup code, at the command line type:

```
tasm C0,C0S.OBJ /D __SMALL__ /D __OLDCONIO__ /MX
```

The startup code object file C0S.OBJ to be linked with your application is created. If you wish to use a unique name for the object file for DeskMate, change C0S.OBJ to the name you want in the command line.

For medium-model startup code, the command entry is:

```
tasm C0,C0M.OBJ /D __MEDIUM__ /D __OLDCONIO__ /MX
```

The medium-model startup object file C0M.OBJ is created.

Linking

When linking your application code with the Turbo startup code and the DeskMate libraries, DM.LIB or DMMED.LIB, you must use the /N option with TLINK.EXE. DM.LIB and DMMED.LIB were created using Microsoft tools, the /N option will tell TLINK not to search for symbols defined in the libraries in the default Turbo libraries.

Debugging under DeskMate

Using Turbo Debugger

If you are compiling using the Microsoft C Compiler:

1. Compile the program using the compile switches:
/Zi - create an object file for use with CodeView debugger
/Od - do not optimize
2. Link the program using the link switches:
/CO - prepare for debugging with CodeView debugger

Note: It is not necessary to compile or link with switches associated with creating line numbers or generating map tables since this is accomplished by running TDCONVRT.EXE.

3. To convert you application linked with Microsoft Link into a format suitable for use with Turbo Debugger, run TDCONVRT.EXE as outlined in your Turbo debugger documentation. The application is now ready for debugging.

If you are compiling using the Turbo C Compiler:

1. Make sure that the startup code for Turbo C 2.0 is set for DeskMate development (see prior Compiling discussion).
2. Compile the program using the compile switch:
-v - create an object file for use with Turbo debugger
3. Link the program using the link switch:
/v - prepare for debugging with Turbo debugger

To run a DeskMate application under Turbo Debugger you must use a setup with a remote machine for the program's output and a local machine to display source code. To debug the application:

1. Rename TDREMOTE.EXE to TDREMOTE.PDM
2. On the remote machine from the DeskTop, run TDREMOTE.PDM.
3. On the local machine (in the same directory as your source code), run TD.EXE with the "-r" (for remote) option.

The Turbo Debugger documentation will describe in detail the sort of messages the remote debugging environment should generate. It is important to note that since screen swapping cannot be used to debug DeskMate applications under Turbo Debugger, the remote method is the only one recommended.

Using SYMDEB

Compile the program using the compile switches:

- /Zd - include line-numbers for source-level debugging
- Od - do not optimize

Link the program using the link switches:

- /LI - uses the line numbers generated by the Zd compile option.
- /M - creates a map table with line numbers.

A symbol file must then be created by using the MAPSYM utility:

```
mapsym yourapp.map
```

Note: It is important that you not run the DESKHDR.EXE extended header utility before debugging the application. Refer to the DeskMate Development Guide, Tools and Utilities section for more information about the header utility.

Now you are ready to enter a debugging session. At the command line, type:

```
symdeb /s <db.in desk.sym yourapp.sym desk.exe
```

The /s option tells symdeb to swap screens. If you are debugging using a remote terminal, do not use this option.

DB.IN is an input file which contains all of the commands and symbol-loading which are needed to get to the break-points in the application. A typical DB.IN file has the following:

```
bp bp_new_task
g
g
xo yourapp! TEXT
z TEXT es+I0
z DGROUP DGROUP+_TEXT
<con
```

The break-point bp_new_task will be encountered twice to load the default application. The DeskTop will appear after the second go, select your application from the DeskTop. The break-point will be encountered when your application is loaded. Load your application's symbol table. Set the code and data segments. Return control to the console.

Now you should be able to examine your code and set any break points you wish, including _main.

Screen swapping does not properly restore color information. If this information is important, you must use a remote terminal for debugging. You may also set your DeskMate screen mode to CGA by using DMVID.EXE with CGA as the mode. This minimizes screen problems using Symdeb.

Using CodeView

Copy CV.EXE to CV.PDM for DeskMate debugging.

Run SETHEAP on CV.PDM with the following switches:

```
SETHelp CV.PDM /MIN 0 /MAX 0
```

Compile the program using the compile switches:

- /Zd - include line-numbers for source-level debugging

- Od - do not optimize

- /Zi - instructs the compiler to include line-number and symbol information in the .OBJ file. You only need to use this option on the modules you wish to debug. Using the /Zd option will include less symbolic information, thus reducing disk space and memory required.

Link the program using the link switches:

- /LI - uses the line numbers generated by the Zd compile option.

- /M - creates a map table with line numbers.

- /CO - for codeview instead of line numbers.

In order to enter a debugging session, it is important that all of the resources needed by the application be pre-loaded in memory. To do this, you must run an application that simply loads and initializes all the necessary resources and exits, leaving the resources in memory. This example loads the GUF resource and the database resource, then exits:

```
main( )
{
    MSGBOX message;

    if ( guf_bind_init() == CSR_ERROR )
        exit( 1 );

    if ( db_bind_init() == CSR_ERROR )
    {
        /* data base couldn't initialize */
        guf_bind_end();
        exit( 1 );
    }

    message.pString = "Resource Loader";
    message.btn_combo = MSG_COMBO_OK;
    message.pMessage = "Resources loaded successfully.";
    msg_run( &message );
    exit( 0 );

} /* end of resource loader */
```


To enter a debugging session in Codeview:

Rename YOURAPP.PDM to YOURAPP.EXE .

Run the DeskMate DeskTop

Run the application above that loads resources and exits (if necessary).

Use the "Run" option from the File menu (F2) to run CV.PDM with the data file name as /s /w YOURAPP.EXE:

/s - use screen swapping

/w - windowing

You may also want to disable the mouse driver during debugging by using the /m option.

Using Periscope

Compile the program using the compile switches:

- /Zd - include line-numbers for source-level debugging
- Od - do not optimize
- /Zi - instructs the compiler to include line-number and symbol information in the .OBJ file. You only need to use this option on the modules you wish to debug. Using the /Zd option will include less symbolic information, thus reducing disk space and memory required.

Link the program using the link switches:

- /LI - uses the line numbers generated by the Zd compile option.
- /M - creates a map table with line numbers.
- /CO - for codeview instead of line numbers.

A symbol file must then be created by using the Periscope TS utility:

```
ts yourapp.map /s /e
```

- /s - Create a symbol file from those in .MAP file
- /e - Read CodeView-able info from executable file for source code display while debugging. Only use this option if the application was compiled with /Zi and linked with /CO.

Now you are ready to enter a debugging session. At the command line, type:

```
run desk.exe
```

RUN instructs Periscope to execute. You must now set your first break-point in Periscope with:

```
>bc bp_new_task
>g
>g
/* select the application from the main menu, or utilize
   the "Run" option from the F2 Menu. */
>ls es+10 yourapp
```

You are now ready to examine code and set further break-points.

Tandy has a set of Periscope macros, developed internally by our programmers, which make DeskMate debugging easier. If you use Periscope for debugging, contact DeskMate Support Services for more information.

DeskMate Coordinate Systems

All video coordinates within the core services are in coordinate points and extents (length). Each is represented in world, device, normalized world, or character coordinates or units. Although most references are in terms of world coordinates and extents, there are a few references to device coordinates and extents and to character extents as parameters for a particular service or a structure element. No core service requires the use of normalized coordinates, however a normalized world coordinate may always be substituted for a world coordinate.

About World Coordinates

A world coordinate maps onto an arbitrary grid of pixels that is usually a much higher resolution than any device which it is intended to represent. Under the DeskMate core, this high resolution grid is defined as 64K x 64K with an origin of (32678, -32768). World coordinate 0,0 is the center of the CSR grid and defaults to the upper-left corner of the currently active window (base window).

With default core definition, the video display surface x extent is 8000 world units and its y extent is 5500 world units. Although the application may change this, it is recommended that the default extents be used when using any of the core user interface services, such as the Dialog Box Manager, the Component Manager, or the Event Manager. Changing the default will scale all origins for core images and text and scale most video images. However, the size of characters and certain images (radio buttons, menu buttons, etc) will not be scaled.

A device coordinate represents the physical device display surface in pixels. Device coordinates range from 0 to the device extent minus 1 (ie., 0-639 x 0-199). All device coordinates outside this range will be clipped. The mapping to a particular device may be redefined through the **vid_set_viewport** call. This viewport call may be used for special scaling, or remapping functions. However, the size of characters and certain images (radio buttons, menu buttons, etc.) will not be scaled.

Under normal usage, several world coordinates refer to the same device coordinate. This is because the number of world coordinates is considerably higher than the number of device coordinates. A **normalized world coordinate** is defined as the world coordinate reference which maps to the upper-left corner of the device pixel. Normalized world coordinates or extents should be used when device pixel accuracy is critical. It is important to note that a world coordinate is not normalized unless it is actually converted to a normalized world coordinate at run time. In other words, on a particular video device a world coordinate which is specified as a constant may be equal to the normalized world coordinate for that pixel. However, that very same coordinate may not be a normalized world coordinate on another device of different resolution. A coordinate can only be considered to be normalized if, at the time of its usage, it is converted to a normalized world coordinate for that particular video device.

The DeskMate core occasionally requires character extents as a unit reference. **Character extents** are character units described in world units.

Using the world screen defaults, the DeskMate screen is defined as 80 characters in the x direction by 25 characters in the y direction. The x extent of a character is CHAR_XEXT world units and the y extent is CHAR_YEXT world units. CHAR_XEXT and CHAR_YEXT are defined in the CSRBASE.INC and CSRBASE.H include file. Changing the core world defaults will change the world unit size of the characters but NOT their physical device unit size.

The origin of the currently active window begins with a default world coordinate 0,0 at the time it is opened. All output which falls outside the clip region (normally the window) will be clipped. Characters which fall only partially within the active window will not appear at all.

Using World Coordinates

The simplest application of world coordinates is positioning on character boundaries. To specify a coordinate which is to represent a character position, multiply the desired character position by CHAR_XEXT or CHAR_YEXT. For example, to move the cursor to character position 40, 12 (approximately the center of the video display surface) use the following method (example in C):

```
vid_move_cursor ( 40 * CHAR_XEXT, 12 * CHAR_YEXT );
```

Remember, if the core default coordinates are changed, the CHAR_XEXT and CHAR_YEXT constants will be invalid.

To specify a coordinate other than on character boundaries, the accuracy of the desired coordinate must be evaluated. Basically, if the desired position is not intended to represent a point immediately adjacent to another point, then constants in world coordinates may be used to specify the coordinate. As an aid to determining these constants, use the character extents to determine the general area and add an offset to that value to specify the position within that character. Keep in mind that the coordinate derived may not give the exact same results from one video device to another.

To specify a coordinate which must be immediately adjacent to or must have a fixed number of device scans between it and another point, a normalized world coordinate must be used. There are four video services provided to perform these calculations for normalized world coordinates.

vid_get_next_nwcx - returns the normalized world coordinate of the next device pixel to the right of a specified world coordinate x.

vid_get_next_nwcy - returns the normalized world coordinate of the next device pixel below a specified world coordinate y.

vid_get_prev_nwcx - returns the normalized world coordinate of the next device pixel to the left of a specified world coordinate x.

vid_get_prev_nwcy - returns the normalized world coordinate of the next device pixel above a specified world coordinate y.

Using World Extents

The number of device pixels that a world coordinate/extent combination encompasses will vary from device to device. If the origin of the extent is normalized then the number of device pixels encompassed by a given extent will always be the same within any single device. However, if the origin and the extent are not normalized, the number of device pixels will vary by one pixel based on whether the origin happens to refer to the normalized world origin for that device or not. Remember, a world coordinate is not normalized unless it has actually been converted to a normalized world coordinate at run time.

This is where the difference between character and graphic output must be considered. All character output is automatically normalized by the core video services. All graphic output is based on the specified world coordinates given as parameters and is NOT automatically normalized. The result of this difference is a difference in the number of device pixels which may

be encompassed by a character output and a graphic output with the same extent. A character will always encompass the same number of device pixels, regardless of the normalization or non-normalization of the origin of the character. A rectangle which has the same extent as a character will encompass a number of device pixels which is dependent on the normalization or non-normalization of the origin of the rectangle. Consider the following example: The string "AB" is output on two different lines of the video display surface, one with a normalized world origin in both x and y, and the other with a non-normalized world origin in both x and y. Also, two rectangles are drawn with the same origins and their x2 and y2 coordinates are calculated from the world extent of a single character. The expected result may be that the rectangle would encompass the same portion of the string on both line. However, because of the above differences in character and graphic output, the string and rectangle output using normalized world origins would result in the rectangle being drawn completely within the character cell of the character "A" in the string. Any portion of the "A" which was at any edge of it's character cell would be overwritten by the rectangle. However, the string and rectangle output using non-normalized world origins would result in the rectangles top and left edges being drawn within the character cell of the "A", the right edge in the character cell of the "B", and the bottom edge on the first device pixel below the character cell of the "A".

World Coordinates - General

World coordinates are a method by which all video screens can be given the same dimensions regardless of the actual pixel resolution of a particular monitor.

First, be sure that you understand the following definitions. The width or height of an item is known as an **extent**. The reference point from which all measurements start is known as an **origin**. The coordinates used as parameters in the routines that draw to the screen are **world coordinates**. The pixel coordinate of an item on your particular monitor is known as a **device coordinate**.

Upon Initialization of the Personal DeskMate core the screen is defined to have a world coordinate origin of (0,0) located in the upper left of the screen. The extent of the screen in the x direction is 8000 world coordinates, while in the y direction the extent is 5500. This can be expressed as follows:

```
wcxOrg = 0
wcyOrg = 0
wcxExt = 8000
wcyExt = 5500
```

A diagonal line can be drawn from the upper left to the lower right with the following: `vid_draw_line(0, 0, 8000, 5500)`

Note that regardless of the resolution of your device, world coordinates instruct the routine to draw a line on the screen that connects the upper left to the lower right. The ability to have the same parameters draw to the same points on any device is known as **device independence**.

Normalized World Coordinates

The world coordinate which lies closest to the upper left corner of a particular device pixel is known as the normalized world coordinate of that pixel. nwc is used to stand for "normalized world coordinate".

When mapping world coordinates to device coordinates it is convenient to visualize a very fine world coordinate grid which overlays a coarse device coordinate screen grid. Each world

coordinate maps to only one device coordinate. But each device coordinate may map to many world coordinates. As an example let us consider a screen that is 640 pixels wide. The world coordinate width is 8000. We can derive the following:

$$w_{cx}Ext = 8000 \quad d_{cx}Ext = 640 \quad 1 \text{ } d_{cx} = 8000/640 = 12.5 \text{ } w_{cx}$$

dcx	nwcx	wcx range	exact nwcx if we had fractions
0	0	0 thru 12	0.0
1	13	13 thru 24	12.5
2	25	25 thru 37	25.0
3	38	38 thru 50	37.5
639	7988	7988 thru 7999	7987.5

Point to Point vs. Origin Extent

vid_draw_line is an example of point to point format. Two points are specified and the line is drawn between the two points.

vid_clear_block is an example of origin extent format. The point of origin is given first, then the extents are relative to that point.

Consider the case where a rectangular block has been cleared to a desired color with the following call:

```
vid_clear_block( 100, 200, 300, 400 )
```

The following will always draw a diagonal line whose endpoints fall exactly on the upper left and lower right corner of the rectangle.

```
vid_draw_line( 100, 200, 399, 599 )
```

The upper left point of the line is the same as the origin of the block. The lower right point is the sum of the origin and extent minus 1. This can be expressed as follows:

```
vid_clear_block( wcxOrg, wcyOrg, wcxExt, wcyExt )
vid_draw_line( wcxOrg, wcyOrg, (wcxOrg+wcxExt-1), (wcyOrg+wcyExt-1) )
```

IMPORTANT: Do not forget the -1 in the line above!!!

Also note that the following 2 lines are equivalent.

```
vid_draw_line( 100, 200, 399, 599 )
vid_draw_line( 399, 599, 100, 200 )
```

However the following 2 lines are NOT equivalent.

```
vid_clear_block( 100, 200, 300, 400 )
vid_clear_block( 300, 400, 100, 200 )
```

Routines that use point to point format start with **vid_draw**.

Origin Independent Extents

The number of pixels that a particular world coordinate extent will span can change depending upon the location of the origin.

To illustrate this consider the following table. A screen width of 640 pixels is assumed. $d_{cx}Ext$ refers to the number of device pixels spanned by $w_{cx}Ext$.

wcxOrg	wcxExt	dcxExt
0	100	8
1	100	9
12	100	9
13	100	8
24	100	9
25	100	8

With `wcxExt = 100` the only cases where `dcxExt` is 8 is when `wcxOrg` is a normalized origin, 100 is the x ext of a character. It turns out that for x extents which are multiples of 100, the `dcxExt` will be constant for any normalized origin. However for x extents which are not multiples of 100 even normalized origins do not assure that the `dcxExt` will be constant.

For example assume a `dcxExt` of 13:

wcxOrg	wcxExt	dcxExt
0	13	1
1	13	2
12	13	2
13	13	2
24	13	2
25	13	1

with `wcxExt = 13` the only cases where `dcxExt` is 1 is when `wcxOrg` maps exactly to a device coordinate with no fraction. In the example above this will occur at even multiples of 25. Thus for the general case, to assure origin independent extents, both the origin and extent need to be normalized. The following is an example.

```
nwcxOrg = vid_wcx_to_nwcx( wxcOrg )
nwcxExt = vid_wcx_to_nwcx( wxcExt-1 ) + 1
```

It turns out that if the origin is on a character boundary then neither the origin or extent need to be normalized to assure origin independent extents.

Finding Adjacent Pixels

If you desire a line to fall on the line adjacent to a given rectangle the world coordinate value of that adjacent pixel must be calculated at run time.

If `wcxOrg` is known to be the origin, the next adjacent pixel to the right of `wcxOrg` is found as follows:

```
nextPixel = vid_next_nwcx( wxcOrg )
```

The following will clear a rectangular block and then draw a rectangle around that block. The rectangle will not cover up any of the block nor will it leave any gaps.

```
vid_clear_block( wxcOrg, wcyOrg, wxcExt, wcyExt )
wxc1 = vid_prev_nwcx( wxcOrg )
wxc1 = vid_prev_nwcy( wcyOrg )
wxc2 = vid_next_nwcx( wxcOrg+wxcExt-1 )
wcy2 = vid_next_nwcy( wcyOrg+wcyExt-1 )
vid_draw_rectangle( wxc1, wcy1, wxc2, wcy2 )
```

Finding the Nth Pixel

A given world coordinate maps to one pixel on the screen. To find the nth pixel relative to that pixel the following method can be used:


```
dcxOrg = vid_wcx_to_dcx( wcxOrg )  
newDcxOrg = dcxOrg + n  
newWcxOrg = vid_dcx_to_wcx( newDcxOrg )
```

n can be a positive or negative integer.
newWcxOrg will map to the nth pixel before or after wcxOrg.

Note that when n = 1 the above method yields the same results as **vid_next_nwcx**, and when n = -1 it yields the same results as **vid_prev_nwcx**.

World, Viewport, Clip

The routines that get or set the world, viewport or clip region are not commonly used by the average application. Before attempting to use these routines please refer to the Window Manager documentation. The Window Manager manages the world, viewport, clip region, and other video states for you. It is strongly suggested that you use the following routines only if you really need the extra versatility they can give you.

The world, viewport, and clip region all work within a 64K x 64K universe. The largest world that can be viewed on the device at one time is a 32K x 32K rectangle within that universe. The universe does wrap around and thus has no edges that the world could bump into.

A world coordinate is a coordinate relative to the origin of the universe. All values used for device or world coordinates are signed integers. Extents are defined to be greater than zero. Origins and point coordinates can be positive or negative. In summary:

Extents = 1 thru 32676 (-32678 thru 0 perform no action)

Origins = -32768 thru 32767

point coordinates = -32768 thru 32767

World and Viewport Relationship

Conceptually the world can normally be thought of as the region of the universe which is mapped to fit onto the screen viewport. The world uses world coordinate units, and is a logical screen. The viewport is measured in pixels or what is called device coordinate units. The viewport is a physical screen.

If the viewport origin is zero then the world origin will map to the upper left corner of the device. If the viewport extent equals the width of the device in pixels, then the world extent will exactly span the width of the device.

However, the viewport origin could be non-zero and the extent could be smaller or bigger than the pixel extent of the screen. The result however is normally undesirable.

Mathematically the mapping equations are:

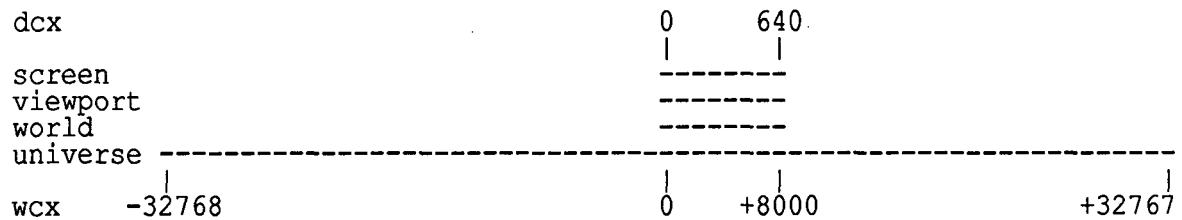
$$dc = (wc - worldOrg) * (viewExt / worldExt) + viewOrg$$
$$wc = (dc - viewOrg) * (worldExt / viewExt) + worldOrg$$

where dc = device coordinate wc = world coordinate

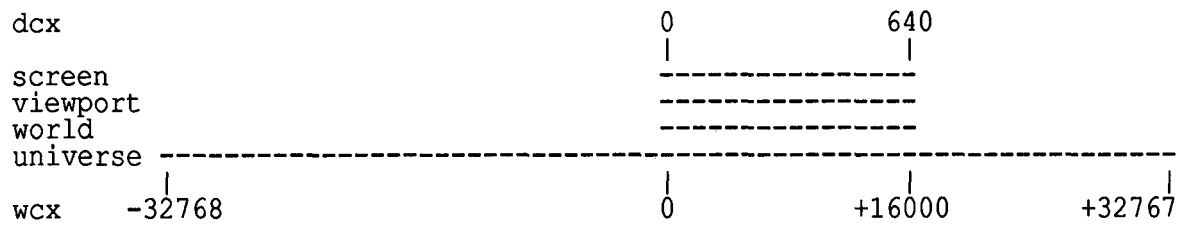
Looking at the first equation it is seen that the scaling of the world to the screen is determined by the ratio of viewExt:worldExt. This ratio controls the squashing necessary to fit the world into the viewport. The worldOrg and viewOrg determine what point of the universe falls onto the upper left point of the screen.

The following examples assume a device that is 640 pixels wide. Only the x direction is dealt with. The y direction would be dealt with in the same manner.

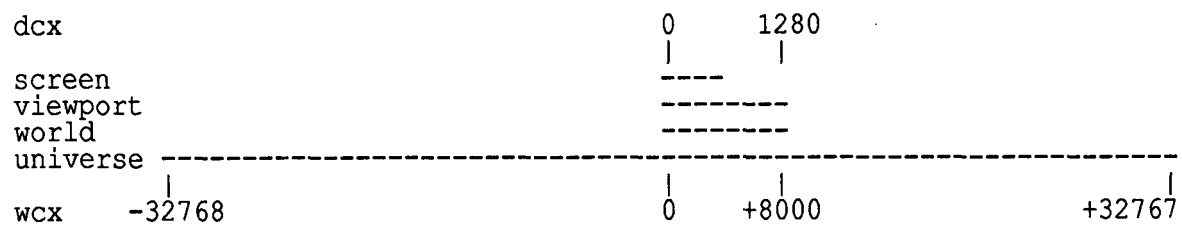
Let: worldOrgX = 0 viewOrgX = 0 worldExtX = 8000 viewExtX = 640



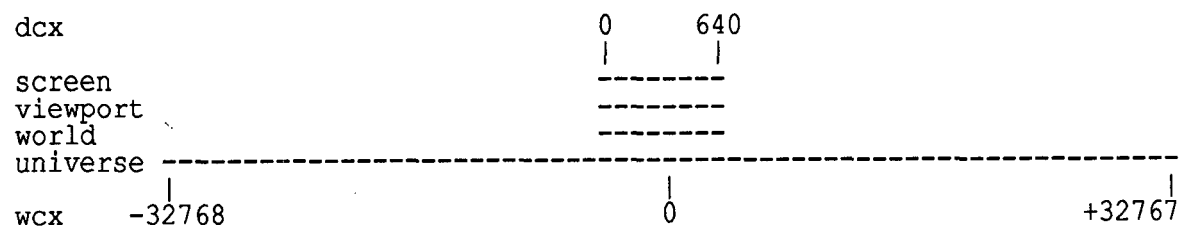
Let: worldOrgX = 0 viewOrgX = 0 worldExtX = 16000 viewExtX = 640



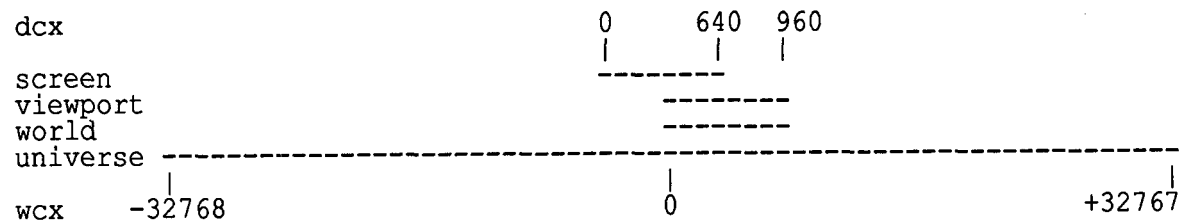
Let: worldOrgX = 0 viewOrgX = 0 worldExtX = 8000 viewExtX = 1280



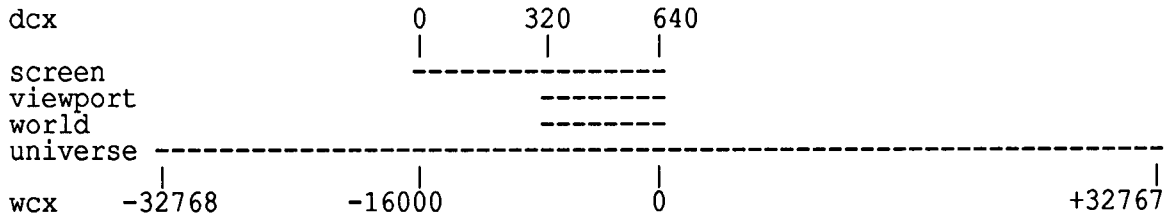
Let: worldOrgX = -4000 viewOrgX = 0 worldExtX = 8000 viewExtX = 640



Let: worldOrgX = 0 viewOrgX = 320 worldExtX = 8000 viewExtX = 640



Let: worldOrgX = -8000 viewOrgX = 320 worldExtX = 8000 viewExtX = 320



As the previous examples illustrate, changing the extent of the world and/or viewport changes the scaling of items mapped to the screen. It is suggested that if changing the scaling is desired, only the world extent should be changed. It normally works out best to leave the extent of the viewport to be equal to the extent of the screen.

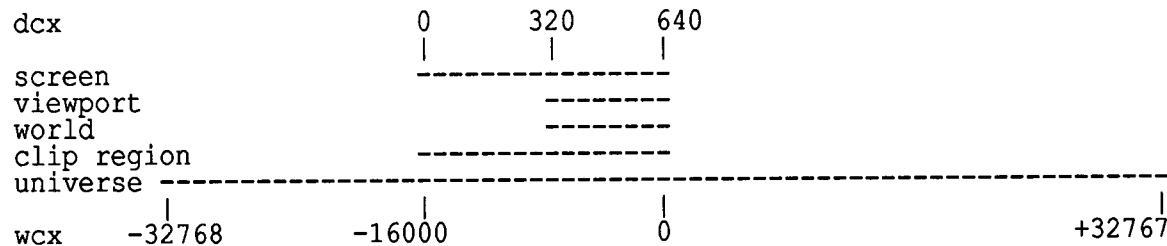
NOTE: Changing the extent of the world to a value other than the defaults of 8000x5500 will change the scaling of routines such as **vid_draw_point**, **vid_draw_line**, **vid_draw_ellipse**, **vid_draw_rectangle**, and stroke fonts, character out routines, and **vid_put_image**. Routines which do not support changes in the scaling are: menu bars, push buttons, radio buttons, check boxes, **vid_clear_to_bot**, **vid_clear_to_eol**, and **vid_put_tty**.

Clip Regions

Clip regions are expressed in world coordinates and are relative to the universe. A clip region with the same origin and extents as the current world will encompass the same region as the current world.

In the example below the screen covers a larger area than the world. A clip region with the same origin and extent as the world would thus allow video routines to write only to pixels 320 to 640. A clip region with an origin of -16000 and extent of 16000 would allow video writes to the entire screen. Note that it is not useful to set the clip region to be larger than the screen, and indeed the **vid_set_clip** routine does not allow you to do so.

Let: worldOrgX = -8000 viewOrgX = 320 worldExtX = 8000 viewExtX = 320
clipOrgX = -16000 clipExtX = 16000



Window Manager

The window manager uses world coordinate origins and extents as input parameters. The origins are positive offsets from the upper left corner of the screen. The extents set a clip region so as to limit video output to the region starting at the window origin and extending thru the window extent. To do this the window manager sets a world and clip region. The viewport is never changed by the window manager. The following is an example:

given: winOrgX, winOrgY, winExtX, winExtY

Compatibility and Programming Issues

Runtime Executive

The 3.3 runtime executive allows an application to be launched from a 3.0 DeskTop as a runtime. This enhancement was added for applications which require the 3.3 environment to operate in but still want to be launched from 3.0 DeskMate products. An application making use of the new 40 column video drivers would be an example of an application requiring the 3.3 system.

To run from a 3.0 DeskTop your application can provide a small "compatibility" application which checks the current system and then runs the application. The compatibility check should also be performed within your application in case the user executes the application from the DeskTop. If your application is large, you should consider providing the compatibility application since it will take less time to load and unload it rather than your application.

The function **dm_compat**, a DeskMate Library function, checks the version of desk currently running and decides if the application

can run on the system.
cannot run because the user is task-switched.
needs to run from the new runtime.

The compatibility application calls **dm_compat**, sending it the name of your customized runtime module, and checks the return code and handles it as follows:

```
main()
{
    int      product_info;
    char      RuntimeName[] = "VENDOR.EXE";

    product_info = dm_compat(&RuntimeName[0]);
    if ((product_info & DM_VERSION) == 0)
    {
        if ((product_info & DM_COMPAT_FLAG) == 0)
        {
            csr_init();
            display "Cannot run while task-switched."
            csr_end();
        }
    } /* running on a 3.0 system */
    else
        /* running on a 3.3 system */
        dm_SetNextApp( to VENDOR.PDM );

    exit();
} /* end of compatibility application */
```

If the application is running on a DeskMate 3.0 system and is not in a task-switched context, then **dm_compat** will call **dm_SetNextApp** to your application's 3.3 runtime. The compatibility application will either cause the application to run on the current system or as a runtime or inform the user that the application cannot run in the current context.

The Help System

DeskMate 3.0 Operation

Help is provided through an accessory. Application help is therefore only available when an accessory can be executed. The application always knows when the user requests help. Applications can write their own event interpreters to capture the F1 key and provide the user with the level of context-sensitive help they deem appropriate.

DeskMate 3.3 Operation

Context-sensitive help is now provided through an Intelligent Help Manager which captures the context of the application and gives specific and general help, specific to the application state. Help is now available in pop-ups, including accessories, and while the menu bar is being accessed. Help may be given at any time, for instance while the user is in a dialog box, and the application is not always aware of when the user requests help. The application can register call-back functions which will be called prior to and after help is given. Refer to the Help Manager section of the Technical Reference for more information.

Compatibility Issues

For applications written for the DeskMate 3.0 system, running on a 3.3 system:

In applications which are not providing any context-sensitive help (by trapping the F1 key), or are not providing help for all the new context possibilities, the user will get a message stating that help is not available. The developer can decide if this is acceptable or do one of following to ensure the user is always presented with help in any DeskMate 3 system.

1) Distribute a Help Compatible System consisting of:

- a) An application help data file.
- b) The help compatibility accessory, DMHELP88.ACC.
- c) The DeskMate 3.3 Intelligent Help Manager, DMHELP.ACC and DMHELPENG.RES.

Upgrade DeskMate 3.0 user's DMHELP.ACC file with the new Intelligent Help Manager, see the Distributing Your Application section in this manual for more information. The new help accessory will chain to the compatibility accessory and provide general application help from the help data file on the upgraded 3.0 system and context-sensitive help on a DeskMate 3.3 system.

2) Handle the new areas of context-sensitive help by using an event interpreter and trapping the F1 key. Refer to the Event Manager section of the Technical Reference for details about writing an event interpreter.

The F10 Tandy Menu

The user can now run new accessories from the More option on the F10 menu or from an upgraded Setup accessory. To run accessories on all DeskMate 3 systems, your application should not perform any range checking on the accessory value before running the accessory. The F10 menu distribution, number of items and their names, varies from

system to system depending on the capabilities of the DeskMate system. Your application should not make any exceptions or assumptions when running accessories, it should simply run the accessory the user requested.

Code Shedding when Running Accessories

DeskMate 3.0 Code Shed Operation

In this environment when an accessory does not fit, the executive code sheds 32K of the application. Applications which can not have their code shed and replaced from disk called `dm_exec_dont_shed`. See the discussion of code shedding in the Introduction of this section for a discussion of code shed criteria.

DeskMate 3.3 Code Shed Operation

In this environment the amount of code shed space for an application is stored in the application's header built by `DESKHDR.EXE`, the DeskMate utility. The executive looks at this information to determine how much, if any, of the application to shed in order to load the accessory. If the code shed size is less than 32K, applications should call `dm_exec_dont_shed` to register that information with the DeskMate 3.0 executive.

Programming and Compatibility Issues

Your application may not function properly if the application cannot be code shed and it does not inform the executive by either setting the code shed size using `DESKHDR.EXE` and/or by calling `dm_exec_dont_shed`.

Your application will not function properly if does its own code shedding to make room for an accessory for the following reasons.

- 1) The DeskMate 3.0 accessories were generally less than 32K, so most accessories would run if that amount of memory was available. In the 3.3 system, most of the accessories use more than 32K. Freeing a specific amount of memory will probably not cover all cases.
- 2) Accessories can load one or more resources when they run. Depending on the function of the accessory, the resource may stay loaded after the accessory exits. For instance, the Spell Checker allows the user to turn on auto-proofing and exit the accessory. The spell resource stays resident to handle the auto-proof function. Your application will not be able to recover the memory it freed for the accessory.
- 3) New accessories may be executed through the new More option, your application cannot predict how these new accessories will operate or how much memory they will require.

If there isn't enough room to load an accessory, the executive will warn the user. It is better not to run an accessory, than to run an accessory and not recover properly.

To run accessories on all DeskMate 3 systems, your application should do the following:

- 1) Set the code shed size (0 up to code size) for your application using DESKHDR.EXE.
- 2) If the code shed size is less than 32K, call **dm_exec_dont_shed** on a DeskMate 3.0 system.
- 3) For applications which use all available memory and cannot be code shed, consider doing one or more of the following:
 - a) shed data which can be regenerated after returning from the accessory.
 - b) shrink the unused data size to free memory for the accessory. Your application must handle not being able to expand out the data if the memory is no longer available.
 - c) free resources which can be reloaded after returning from the accessory. Your application must handle not being able to reload the resources if the memory is no longer available.

"Sticky Menus" and Selectable Grayed Menu Items

Since the menu bar processing is done within the DeskMate environment, this enhancement is transparent to the application. Applications which use their own event interpreters and are predicting the state of the menu bar based on the mouse or arrow events are affected by this change.

In the DeskMate 3.0 system, a single mouse click did not affect the state of a menu bar. In the 3.3 system, a single mouse click can cause a menu to drop or will change the selection of a menu item.

In the 3.0 system, the up and down arrows skipped over grayed menu items. In the 3.3 system, the up and down arrows do not skip grayed menu items.

To be compatible on all DeskMate 3 systems, applications which predict user events must handle the differences in the menu bar user interface in each system. To aid the developer, the new **mb_get_status** call was added to get menu bar status information.

Animated Busy Icon

The Tandy busy icon is now animated. The icon processing can cause problems for applications which are accessing video memory directly and are making timing assumptions about the busy icon. If your application meets this criteria, make sure your application disables the busy icon while it is accessing video memory.

Form Manager and GUF Resource

Loading of the Resources for 3.0 Applications

The DMFORM.RES is automatically loaded on the first **form_open** call. Both GUF resources, DMGUF .R89 and PRGUF .RES are loaded with the **guf_bind_init** call.

If the resource does not fit in available memory or the resource file could not be found, the **form_open** and **guf_bind_init** calls will return an error. You should ensure your application is checking the return code from both call and handles the conditions properly.

If your application uses all available memory, the **form_open** call should be made BEFORE all of memory is allocated.

Loading of the Resources for 3.3 Applications

The new binding call for the Form Manager resource, **csr_form_bind_init** will return an informative error DM_EXISTS if the application is running on a 3.0 system.

Both GUF resources, DMGUF .R89 and PRGUF .RES are loaded with the **guf_bind_init** call. To load only the PRGUF .RES resource, call **prguf_bind_init**.

Video Drivers

Driver Names

The DeskMate 3.0 video drivers used the DMVD prefix, the 3.3 drivers use the DMVS prefix. The video drivers must match the version of the CSR being used, mixing of systems is not allowed. Applications using the **cfg_get_vid_driver** call to determine what video driver is loaded are affected by this change and should handle the differences in the systems.

Video Detection

The VGA video driver, DMVDVGA.RES, incorrectly returned VID_EGA in the VID_DEVICE.card element when the **vid_inquire_device** call was made. In order to determine if the video was in fact VGA, the calling program compared the VID_DEVICE.dc_yext element to 480. The VGA video driver, DMVSVGA.RES, correctly returns VID_VGA from the **vid_inquire_device** call. If your application makes use of the **vid_inquire_device** call, you should ensure you handle the differences appropriately.

Palettes

The DMVSVGA driver uses different palettes than those used by the DMVDVGA driver. If your application accesses the palette information directly, then your application will exhibit different default color settings in the 3.0 and 3.3 environments.

Printer Drivers

Line Styles

The line widths, LINE_WIDTH1 and LINE_WIDTH2 are now supported for the dotted, dashed, and dot-dash line styles. These widths were only supported for LINE_WIDTH1 which exhibited printing problems when a line crossed a print band.

The line style DENSE_DASHED is now supported by the printer drivers.

The thickness of the wider line widths was changed to match the world coordinate width used by the video drivers.

LINE_WIDTH1	1 pixel wide
LINE_WIDTH2	"best look", normally 2 pixels wide
LINE_WIDTH3	50 world coordinates wide
LINE_WIDTH4	75 world coordinates wide
LINE_WIDTH5	100 world coordinates wide

Print regions

The 132 character maximum line has been removed and now as many characters as will fit into the width of the print band will be printed. The width of the print band for printers with a wide carriage is 13200 world coordinates. This translates to the following number of characters depending on the current character per inch setting:

10 CPI	132 characters
12 CPI	158 characters
condensed	220 characters

The dimensions of the printable region for the 3.0 printer drivers was sometimes less than 8 x 11 1/2 inches. The 3.3 printer drivers now print exactly to 8 x 11 1/2 inches. This apply to IBM-compatible graphics printers. The Tandy 2100P with micro line-feed control prints a page 11 3/8 inches instead of 11 1/2. Other non-Tandy printers exhibit the same incompatibility.

The quarter-inch on the left and right side of the paper is the default "unprintable region" for printers. The laser printer has its own specific unprintable region.

Landscape printing

The DeskMate 3.0 drivers did not do a form feed at the end of a landscape printed page, the new drivers do.

Overview of the Tools, Utilities, and Examples

The tools and utilities included in the kit are provided as an aid to the programmer. Most of the tools have not had the software quality assurance testing the DeskMate products and system files have had. Tandy uses these tools internally for the development of the DeskMate product and system files. We do not warrant these tools and utilities and recommend that you take the precaution of backing up your work files when using the tools. User documentation on using the tools is provided in the Tools and Utilities section of this manual.

The Menu Bar Builder and Dialog Box Builder are used to build the major portion of your application's user interface are provided as time saving tools. You do not have to use these tools to build either your menu bar or dialog box data structure definitions. Both of these tools have been improved and we encourage that you use these tools.

We provide several graphics utilities - the Bitmap Editor, the Graphics Form Generator, the Clipart File Builder, and the Stroke Font Editor. These tools are used to either import graphics into your application or customize graphics files used by your application. The tools have varying degrees of functionality.

The Memory Map Generator is used to determine how memory is being allocated (size and distribution) under DeskMate. With all of the different environment possibilities, this tool is very useful for determining how much memory your application has to work with and how it is distributed. This tool allows you to determine the worst and best possible cases under which your application will run.

The Desk Header Builder is used by all of our DeskMate product and system files. We strongly recommend every developer use it to build the DeskMate application's file header.

The Disk Label Generator is used to build the diskette label files for all of the DeskMate 3 products. We strongly recommend using it to create diskette label files for your product also. These labels are used by `dm_file_search` when prompting for a diskette and may be used by your installation program.

The Customized Runtime Utility **MUST** be used to build your application's customized executive.

The Customized Install Utility is used to build your application's customized version of `INSTALL.EXE` which must accompany your product diskette. This program launches your `INSTALL.PDM` program as a stand-alone program using your customized runtime executive.

The Help Utilities must be used to create your application's help file if you make use of the DeskMate Intelligent Help Manager. Tandy used these tools to create the help files distributed with the DeskMate 3.3 products.

The Tutorial Technology Tools are used to build application tutorials and demos. The `DMEI.EXE`, `RECORD.PDM`, and `DMRECORD.RES` tools are only needed if you choose to "record" your initial tutorial or demo script. The other tutorial tools are needed to build tutorial files for execution by the Play technology.

The sample programs include a Welcome program which is an example of a minimal DeskMate application. The Video sample shows how to use world and device coordinates when displaying graphics on the screen. This example also includes a sample help file, `VIDEO.HLP`.

The File I/O and Printing examples are quite extensive. The low-level examples of both the file i/o and printing actually reflect much of the processing done by GUF and the Print Manager in the high-level examples. The High-Level File I/O example is a good start for an application user interface prototype which uses canned data files. Once your data requirements are established you can rewrite the file i/o portion of your application.

The Forms example shows the basic functions performed by the Form Manager for managing graphics in an application. The full power of the Form Manager is not demonstrated by this example. It is a good starting point and can be used as a template for graphics handling applications.

The Special Topics section discusses some programming techniques which are part of the other examples but were not discussed in the sample chapters, for instance Interfacing With the Clipboard discusses the edit field functions used by the COMPS sample and the form manager functions used by the FORMS sample that interface with the clipboard. Programming excerpts are used to discuss Managing Multiple Windows and Events. The COMPS example shows how to manage components in the work area instead of in dialog boxes as a means of interfacing with the user.

The last three sections discuss the special programming requirements of 40 column applications, DeskMate resources, and DeskMate accessories.

Part 2
Programming Examples

Contents

A DeskMate Shell - WELCOME.PDM	2-1
Using the DeskMate Coordinate Systems - VIDEO.PDM	2-7
DeskMate File Handling	2-13
High-Level File I/O - FILEIOHL.PDM	2-13
Low-Level File I/O - FILEIOLL.PDM	2-19
Database File I/O - DBCARS.PDM	2-31
Printing	2-47
Page Printing - DEVICE.PDM	2-47
Direct Printing - DIRECT.PDM	2-53
Using the Graphics Form Manager - FORMS.PDM	2-57
Special Topics	2-65
Running Components in the Work Area - COMPS.PDM	2-65
Managing Windows and Events	2-69
Interfacing with the Clipboard	2-71
From an Editfield Component	2-71
When Using the Form Manager	2-71
Direct Interfacing with the Clipboard	2-75
To read the clipboard	2-75
To write to the clipboard	2-75
Writing text with attributes to the clipboard	2-76
Writing a 40 Column Application	2-77
Writing a DeskMate Resource	2-79
Writing a DeskMate Accessory	2-89
General Guidelines	2-89
Accessory Chaining	2-90

WELCOME.PDM

WELCOME.PDM is an example of a minimal DeskMate application. It has an application menu bar, places text in the work area, uses the About function, runs other applications through the File Run option, and Exits. It is ideal for use as a template for beginning new DeskMate applications. The source to the Welcome application is included in the SAMPLES\WELCOME directory.

```
#include "dmexec.h"      /* Desk Executive header file */
#include "csrbase.h"     /* Core Services Resource base header file */
#include "csrcmps.h"     /* Core Services Resource components header file */
#include "csrvid.h"      /* Core Services Resource video header file */
#include "dmdecl.h"      /* DeskMate function declarations */
#include "dmguf.h"       /* General User Functions */
#include "dmgufdec.h"    /* GUF's function declarations */
#include "codestnd.h"    /* DeskMate Coding standards file */
#include "welcome.h"     /* Application header file */
#include "welcodec.h"    /* Application declarations */
```

The Welcome application first includes the DeskMate header files it requires to compile. These files reside on the Development Diskette. The following is a brief description of each file and why it is included in this example application.

The DMEEXEC.H file is the include file for the Desk Executive. All applications should include this header file.

The CSRBASE.H file should also be included by all applications. This file is the standard include file for the Core Services Resource. It defines many of the general structures used by applications.

The CSRCMPS.H file contains the component data structures, including the menu bar definitions. Source files accessing component, menubar, dialog box, and message box data structures require this file.

The CSRVID.H file includes many of the structures and defines needed when accessing the video functions of the CSR.

The DMDECL.H, and WELCODEC.H are function prototype files. These files are used by the compiler to check the parameter(s) being passed to functions defined in the files. Using these files often catches programming problems related to errors in calling the function at the compile stage.

The DMGUF.H file is the standard include file for the DMGUF and PRGUF resources. Applications using these resources to perform File I/O, use the Environment Manager, or support the Run command need to include this file in their applications.

The CODESTND.H file describes the DeskMate coding standards used in these sample applications. This file is for information purposes only.

The WELCOME.H file contains the data structures and defines used by this application.


```

int main()
{
    EVENT Event;
    int    TSReturnCode;
    int    Done;

    /* Bind to the Core Services Resource */
    if ( csr_init() == CSR_ERROR )
        /* failure to bind to the CSR, could not find/load resource */
        exit(1);

    if ( guf_bind_init() == CSR_ERROR )
    {
        /* failure to bind to the power & run General User Functions resource */
        csr_end();
        exit(1);
    }
}

```

Before any processing is done by your application, your application must bind to the resources it will use during its execution. Notice that both binding calls check the return code. Although the CSR is almost always guaranteed to be loaded when your application is loaded, it is good programming practice to check the `csr_init`'s, as well as all bind routine, return codes.

Your application should never make assumptions about the state of environment when your application is loaded. For more information about resource binding routines, see the **Desk Executive** section of the DeskMate Technical Reference.

```

/* Draw the main screen */
Welcome_Draw_Screen();

```

The next step in an application's processing is usually to draw the application menu bar and default work area. This sample calls a subroutine `Welcome_Draw_Screen` to perform the task.

```

void Welcome_Draw_Screen()
{
    /* Clear the base window (defaults to the entire screen) */
    vid_clear_screen();

    /* Draw the application menubar in the base window */
    WelcomeMENUBAR.bRedraw = MB REDRAW;
    mb_draw( &WelcomeMENUBAR );

    /* Display the application's name on the title line */
    ttl_put_app_name( "Welcome" );

    /* Display the application's data file name on the title line */
    ttl_put_data_name( "" );

    Welcome_Draw_Message();
}

```

`Welcome`'s draw screen processing, clears the entire screen, draws the application menu bar, displays the application name, displays "(Untitled)" for the datafile name, and displays a welcome message in the work area.

```

void Welcome_Draw_Message()
{
    /* Position the video cursor in the center of the base window */
    vid_move_cursor( 30 * CHAR_XEXT, 12 * CHAR_YEXT );

    /* Display "Welcome to DeskMate!" in the center of the window */
    vid_put_string( "Welcome To DeskMate!" );

    /* Set the line color attributes to make sure when the */
    /* rectangle below is drawn, it shows up on all videos */
    vid_set_line_attr( LINE_SOLID, LINE_WIDTH1, COLOR3 );

    /* draw a rectangle in the center of the base window */
    vid_draw_rect( 29 * CHAR_XEXT, 11 * CHAR_YEXT,
                  51 * CHAR_XEXT, 14 * CHAR_YEXT,
                  VID_NO_FILL );
}

```


The welcome message displayed in the work area uses video calls to reposition the cursor (default location 0,0), display a text string at the current cursor location, and draw a box around the text.

For more information about the function calls used here, see the **Video Manager** (vid_*), **Menu Bar Manager** (mb_*), and **Titleline Manager** (ttl_*) sections of the DeskMate Technical Reference.

Now the application is ready to accept user input. DeskMate applications are usually written as transaction centers (all of our samples are). The application executes in a loop until the user chooses to exit the application (or an error condition requiring termination is encountered). In the loop processing the application waits for a user event and then acts upon that event, usually passing it on to a module which will process the event.

```
/* initialize the do while control flag */
Done = FALSE;

/* Process the user inputs and actions */
do
{
    /* read an event from an input device */
    event_read( &Event );
```

The **event_read** call will not return until the user performs an action which is translated into an event. For example, simply moving the mouse around the screen will not return an event. When the user presses the mouse button, the function will return with the appropriate event in the **EVENT** structure.

Command Events, **EVENT_COMMAND**, are returned whenever the user selects a menu option or a component in the work area. The **EVENT.param** element contains the value assigned to the return code element of the **MENUITEM** structure for each menu option or in the **CMP_HEADER** structure for a component.

```
switch( Event.msg )
{
    case EVENT_COMMAND:
        /* check to see if an option was selected from the menu bar */
        /* process menu option that was selected */

        switch( Event.param )
        {
            case FILE_EXIT_ID:
                /* the user wants to exit */
                Done = TRUE;
                break;
```

Event param is return code of component.

DeskMate applications should include the Run menu option and should use either the **fil_menu_run** or **dlgbox_Run** calls to display the Run File dialog box. For more information about the **fil_menu_run** function, see the File I/O Examples in this section.

```
case FILE_RUN_ID:
    /* the user wants to exit, then run another program */
    if( dlgbox_Run() == TRUE )
        Done = TRUE;

    /* clear the dlgbox Run dialog box from the screen */
    vid_move_cursor( 0 * CHAR_XEXT, 7 * CHAR_YEXT );
    vid_clear_to_bot();

    /* display the welcome message back on the screen */
    Welcome_Draw_Message();
    break;
```


Every DeskMate application should include the About menu option and should use the **about_versions** call to display the information about the application. This function provides information about all applications currently in memory along with the application. This information is useful when customers call with a problem.

```

case FILE ABOUT_ID:
    /* make the DeskMate library call to */
    /* display an ABOUT... dialog box */
    about_versions( &WelcomeAPPL_VERSION );

    /* clear the dialog box from the screen */
    vid_move_cursor( 0 * CHAR_XEXT, 3 * CHAR_YEXT );
    vid_clear_to_bot();

    /* display the welcome message back up on the screen */
    Welcome_Draw_Message();
    break;

default:
    break;

} /* end of switch on type of application event */
break;

```

Application Events, **EVENT_APPL**, cause the application to turn over control to another process. Applications which access the data file during operations should ensure that the file is on the disk when running on a floppy system. The task switch or accessory may have caused the user to swap disks. Make sure your data disk is in the driver before writing to the file.

The **APPL_ACCESS** event is returned when the user chooses an accessory option from the F10 Accessories Menu. The **dm_acc_run** call is used to actually run the accessory. Applications should always redraw their entire screen (and check the clipboard if they have an Edit Menu) when returning from running an accessory.

```

case EVENT APPL :
    switch( Event.param )
    {
        /* check for an accessory event */
        case APPL_ACCESS:
            /* run the requested accessory */
            dm_acc_run ( Event.x );

            /* redraw the screen when the accessory is finished */
            Welcome_Draw_Screen();
            break;
    }

```

The **APPL_TASK_SWITCH** event is returned when the user chooses the Task Switch option from the F10 Menu or uses the ALT+= accelerator. The **dm_yield** call is used to do the actual task switch. Task switching is not allowed in all environments, the application must check the return code and handle the return code appropriately as specified here.

Applications which use the Clipboard should check the Clipboard contents and enable Paste if necessary. Applications using Page Setup should set the Page Setup information after a task switch since the other application may have changed it.

```

/* check for a task switch event */
case APPL_TASK_SWITCH:
    /* attempt to execute a task switch */
    TSReturnCode = dm_yield();

    if ( TSReturnCode == DM_NOT_ALLOWED )
        /* task switching not being allowed */
        break;

    if ( TSReturnCode == DM_OK )
        /* The task switch has occurred so */
        /* redraw the menubar and the screen */
        Welcome_Draw_Screen();

```



```

        else
        {
            /* the yield (task switch) failed */
            /* there is bad trouble so exit program */
            Done = TRUE;
        }
        break;

    default:
        break;
} /* end of switch on type of Command */
break;

default:
    break;

} /* end of switch on type of event */

}
/* check to see if "EXIT" or "RUN" menu option has been selected */
while( Done != TRUE );

```

As you can see, even a simple application requires the processing of several types of user events or actions during its execution.

```

/* inform the loaded resources that the application is exiting */
guf_bind_end();
csr_end();
exit(0);
}

```

The last step before the application terminates is the freeing of the resources the application used during its execution. The resources should be freed in the reverse order of the way they were requested. The application then exits. Applications launched from the DeskTop will return the user to the DeskTop when they exit. Applications launched from a runtime executive will return the user to the DOS prompt when they exit.

VIDEO.PDM

VIDEO.PDM is a slightly more advanced example of a DeskMate application. It introduces the concept of device and world coordinates systems which DeskMate uses. The source to the Video application is included in the SAMPLES\VIDEO directory.

```
#include "csrkeys.h"    /* Core Services Resource keyboard header file */
#include "video.h"       /* Application header file */
#include "videodec.h"    /* Application's function declarations */
```

The Video application includes the DeskMate header files it requires to compile, these files are described in the preceding example, Welcome. The following is a brief description of the additional files this application uses and why they are included in this example application.

The CSRKEYS.H file contains the key defines used in DeskMate.

The VIDEO.H file contains the data structures and defines used by this application.

The VIDEODEC.H file contains the function prototypes for functions in this application.

Like the Welcome application, this application first binds to the resources it will use. Before drawing the menu bar, Video sets the menus to their default configuration. It sets all of the options in the Shapes Menu to an unchecked state and then checks the rectangle option.

```
/* check the first shapes menu item */
Video_Uncheck_Shapes_Menu();
VideoShapesMENUITEM[RECT_INDEX].bChecked = MB_CHECKED;

void Video_Uncheck_Shapes_Menu()
{
    int i;

    for( i = 0; i < NUM_SHAPES_MENU_ITEMS; i++ )
        VideoShapesMENUITEM[i].bChecked = MB_UNCHECKED;
}
```

The Video_Draw_Screen subroutine performs the same functions as Welcome's draw screen routine.

```
/* Draw the main screen */
Video_Draw_Screen();
```

This example uses a routine to draw objects in the work area. Notice that the routine is called to draw a "default" object on the screen before the user does anything.

```
/* Draw the first "default" object */
Video_Draw_Object();
```

Several shapes are drawn by the routine, a rectangle, ellipse, line, and point. Each of these graphics shapes as well as others are supported by the Video Manager in the CSR. All of the functions use the world coordinate system to specify the location and size of the graphics object.

```
void Video_Draw_Object()
{
    int ExtX = CurrentX;
    int ExtY = CurrentY;

    /* Set the line color attributes to make sure when the */
    /* object below is drawn, it shows up on all videos */
    vid_set_line_attr( LINE_SOLID, LINE_WIDTH1, COLOR3 );
```



```

switch( ShapeSwitch )
{
    case SHAPE_RECT ID:
        ExtX += 5 * CHAR_XEXT;
        ExtY += 1 * CHAR_YEXT;
        vid_draw_rect( CurrentX, CurrentY, ExtX, ExtY, VID_NO_FILL );
        break;

    case SHAPE_ELLIPSE ID:
        vid_draw_ellipse( CurrentX, CurrentY, 2 * CHAR_XEXT, 1 * CHAR_YEXT,
                          VID_NO_FILL );
        break;

    case SHAPE_LINE ID:
        ExtX += 5 * CHAR_XEXT;
        ExtY += 1 * CHAR_YEXT;
        vid_draw_line( CurrentX, CurrentY, ExtX, ExtY );
        break;

    case SHAPE_POINT ID:
        vid_draw_point( CurrentX, CurrentY );
        break;
}

Video_Draw_Coordinates();
}

```

The status area at the bottom of the work area shows the values of the Current X and Y location of the cursor on the screen in World Coordinates. Notice that `sprintf` is used to build the display string and that the actual displaying of the text is done with `vid_put_string`. Some video modes, Hercules for instance, do not support text operations when in graphics mode. Because DeskMate applications operate in a graphics mode, text out operations need to be done through the video manager to guarantee the text will appear in all video modes.

```

void Video_Draw_Coordinates()
{
    char    TempBuf[60];

    /* Display the value for CurrentX & CurrentY */
    vid_move_cursor( 0 * CHAR_XEXT, 24 * CHAR_YEXT );
    sprintf(TempBuf, "World Coordinate X = %d", CurrentX );
    vid_put_string( TempBuf );
    vid_put_string( " "); /* clear out old number */
    vid_move_cursor( 30 * CHAR_XEXT, 24 * CHAR_YEXT );
    sprintf(TempBuf, "World Coordinate Y = %d", CurrentY );
    vid_put_string( TempBuf );
    vid_put_string( " "); /* clear out old number */
}

```

Like Welcome, this application also uses an event loop to do its processing. Only the events specific to this example are discussed here. Notice, this example was built by using the Welcome template.

```

switch( Event.param )
{
    case FILE_UPDATE ID:
        /* redraw the screen at the users request */
        Video_Draw_Screen();
        Video_Draw_Object();
        break;
}

```

These are commands received from the user. Adding to and subtracting from the current X any Y world coordinate.

```

/* 13 world coordinates = 1 pixel */
case WORLD_NEXTX ID:
    CurrentX += 1;
    Video_Draw_Object();
    break;

```



```

case WORLD NEXTY ID:
    CurrentY += I;
    Video_Draw_Object();
    break;

case WORLD PREVX ID:
    CurrentX -= I;
    Video_Draw_Object();
    break;

case WORLD PREVY ID:
    CurrentY -= I;
    Video_Draw_Object();
    break;

```

This modifies header information in the edit field so that 4 characters are allowed to be entered. After the dialog box is run the number entered in the edit field is added to or subtracted from the Current X or Y and the current object is drawn at the newly specified location.

```

case WORLD NEXTNX ID:
    VideoFRAME[0].pString = VideoWorldNextNXStr;
    VideoEDITFIELD.header.maprect.xext = (4 * CHAR XEXT);
    VideoEDITFIELD.edit maprect.xext = (4 * CHAR XEXT);
    VideoEDITFIELD.pBuffer = VideoWorldEDITFIELDDBuff;
    DLGReturnCode = Video_Process_Dialog();
    if( DLGReturnCode == VideoOKtag )
    {
        CurrentX += atoi( VideoWorldEDITFIELDDBuff );
        Video_Draw_Object();
    }
    break;

case WORLD NEXTNY ID:
    VideoFRAME[0].pString = VideoWorldNextNYStr;
    VideoEDITFIELD.header.maprect.xext = (4 * CHAR XEXT);
    VideoEDITFIELD.edit maprect.xext = (4 * CHAR XEXT);
    VideoEDITFIELD.pBuffer = VideoWorldEDITFIELDDBuff;
    DLGReturnCode = Video_Process_Dialog();
    if( DLGReturnCode == VideoOKtag )
    {
        CurrentY += atoi( VideoWorldEDITFIELDDBuff );
        Video_Draw_Object();
    }
    break;

case WORLD PREVNx ID:
    VideoFRAME[0].pString = VideoWorldPrevNXStr;
    VideoEDITFIELD.header.maprect.xext = (4 * CHAR XEXT);
    VideoEDITFIELD.edit maprect.xext = (4 * CHAR XEXT);
    VideoEDITFIELD.pBuffer = VideoWorldEDITFIELDDBuff;
    DLGReturnCode = Video_Process_Dialog();
    if( DLGReturnCode == VideoOKtag )
    {
        CurrentX -= atoi( VideoWorldEDITFIELDDBuff );
        Video_Draw_Object();
    }
    break;

case WORLD PREVNY ID:
    VideoFRAME[0].pString = VideoWorldPrevNYStr;
    VideoEDITFIELD.header.maprect.xext = (4 * CHAR XEXT);
    VideoEDITFIELD.edit maprect.xext = (4 * CHAR XEXT);
    VideoEDITFIELD.pBuffer = VideoWorldEDITFIELDDBuff;
    DLGReturnCode = Video_Process_Dialog();
    if( DLGReturnCode == VideoOKtag )
    {
        CurrentY -= atoi( VideoWorldEDITFIELDDBuff );
        Video_Draw_Object();
    }
    break;

```

The device commands use the video routines to get the next and previous world coordinates, then displays the object.

```

case DEVICE NEXTX ID:
    CurrentX = vid_next_nwcx( CurrentX );
    Video_Draw_Object();
    break;

```



```

case DEVICE NEXTY ID:
    CurrentY = vid_next_nwcy( CurrentY );
    Video_Draw_Object();
    break;

case DEVICE PREVX ID:
    CurrentX = vid_prev_nwcx( CurrentX );
    Video_Draw_Object();
    break;

case DEVICE PREVY ID:
    CurrentY = vid_prev_nwcy( CurrentY );
    Video_Draw_Object();
    break;

```

These modify the header info in the edit field of the dialog box so that only 3 characters are allowed to be entered. After the dialog box is run successfully the number the user entered is again sent to the get prev and next world coordinate video routines then the object is displayed.

```

case DEVICE NEXTNX ID:
    VideoFRAME[0].pString = VideoDeviceNextNXStr;
    VideoEDITFIELD.header.maprect.xext = (3 * CHAR XEXT);
    VideoEDITFIELD.edit maprect.xext = (3 * CHAR XEXT);
    VideoEDITFIELD.pBuffer = VideoDeviceEDITFIELDDBuff;
    DLGReturnCode = Video_Process_Dialog();
    if( DLGReturnCode == VideoOKtag )
    {
        CurrentX = vid_nextn_nwcx( CurrentX,
                                   atoi(VideoDeviceEDITFIELDDBuff) );
        Video_Draw_Object();
    }
    break;

case DEVICE NEXTNY ID:
    VideoFRAME[0].pString = VideoDeviceNextNYStr;
    VideoEDITFIELD.header.maprect.xext = (3 * CHAR XEXT);
    VideoEDITFIELD.edit maprect.xext = (3 * CHAR XEXT);
    VideoEDITFIELD.pBuffer = VideoDeviceEDITFIELDDBuff;
    DLGReturnCode = Video_Process_Dialog();
    if( DLGReturnCode == VideoOKtag )
    {
        CurrentY = vid_nextn_nwcy( CurrentY,
                                   atoi(VideoDeviceEDITFIELDDBuff) );
        Video_Draw_Object();
    }
    break;

case DEVICE PREVNX ID:
    VideoFRAME[0].pString = VideoDevicePrevNXStr;
    VideoEDITFIELD.header.maprect.xext = (3 * CHAR XEXT);
    VideoEDITFIELD.edit maprect.xext = (3 * CHAR XEXT);
    VideoEDITFIELD.pBuffer = VideoDeviceEDITFIELDDBuff;
    DLGReturnCode = Video_Process_Dialog();
    if( DLGReturnCode == VideoOKtag )
    {
        CurrentX = vid_prevn_nwcx( CurrentX,
                                   atoi(VideoDeviceEDITFIELDDBuff) );
        Video_Draw_Object();
    }
    break;

case DEVICE PREVNY ID:
    VideoFRAME[0].pString = VideoDevicePrevNYStr;
    VideoEDITFIELD.header.maprect.xext = (3 * CHAR XEXT);
    VideoEDITFIELD.edit maprect.xext = (3 * CHAR XEXT);
    VideoEDITFIELD.pBuffer = VideoDeviceEDITFIELDDBuff;
    DLGReturnCode = Video_Process_Dialog();
    if( DLGReturnCode == VideoOKtag )
    {
        CurrentY = vid_prevn_nwcy( CurrentY,
                                   atoi(VideoDeviceEDITFIELDDBuff) );
        Video_Draw_Object();
    }
    break;

```


These commands uncheck all items in the shapes menu, then check the appropriate one.

```

case SHAPE_RECT_ID:
    Video_Uncheck_Shapes_Menu();
    VideoShapesMENUITEM[RECT_INDEX].bChecked
                                = MB_CHECKED;
    ShapeSwitch = SHAPE_RECT_ID;
    Video_Draw_Object();
    break;

case SHAPE_ELLIPSE_ID:
    Video_Uncheck_Shapes_Menu();
    VideoShapesMENUITEM[ELLIPSE_INDEX].bChecked
                                = MB_CHECKED;
    ShapeSwitch = SHAPE_ELLIPSE_ID;
    Video_Draw_Object();
    break;

case SHAPE_LINE_ID:
    Video_Uncheck_Shapes_Menu();
    VideoShapesMENUITEM[LINE_INDEX].bChecked
                                = MB_CHECKED;
    ShapeSwitch = SHAPE_LINE_ID;
    Video_Draw_Object();
    break;

case SHAPE_POINT_ID:
    Video_Uncheck_Shapes_Menu();
    VideoShapesMENUITEM[POINT_INDEX].bChecked
                                = MB_CHECKED;
    ShapeSwitch = SHAPE_POINT_ID;
    Video_Draw_Object();
    break;

default:
    break;

} /* end of switch on type of application event */
break;

```

This routine, Video_Process_Dialog processes a dialog box. The first thing it does is pull up the push buttons (for subsequent run) and initialize the focus index (defines which component should have the focus when the dialog box is run). The cursor offset is set so that any data in the edit field will be selected. (Safeguard requirement). It then attempts to save the part of screen that will be covered by the dialog box. Draws and runs the dialog box, until OK or CANCEL is returned. Then redisplay the given portion that was saved, or redraws the entire screen, then returns the OK or CANCEL return to the calling routine.

```

int Video_Process_Dialog()
{
    unsigned int    DLGReturnCode;
    unsigned int    BufferSize;
    char            *pBufferSize;
    int             RedrawFlag;

    VideoPUSHBUTTON[0].bState = PB_UP;
    VideoPUSHBUTTON[1].bState = PB_UP;
    VideoDIALOG_BOX.focus_index = 0;
    VideoEDITFIELD.cursor_offset = EF_SELECT_ALL;
    BufferSize = vid_get_buffer_size(
        VideoFRAME[0].maprect.xorg - (2 * CHAR_XEXT),
        VideoFRAME[0].maprect.yorg - (2 * CHAR_YEXT),
        VideoFRAME[0].maprect.xext + (3 * CHAR_XEXT),
        VideoFRAME[0].maprect.yext + (3 * CHAR_YEXT) );
    if( BufferSize == CSR_ERROR )
        RedrawFlag = TRUE;
    else
        RedrawFlag = FALSE;
    pBufferSize = (char *)malloc( BufferSize );
    if( pBufferSize == (char *)0 )
        RedrawFlag = TRUE;
    else
        RedrawFlag = FALSE;
}

```



```

vid_get_screen( VideoFRAME[0].maprect.xorg - (2 * CHAR_XEXT),
                VideoFRAME[0].maprect.yorg - (2 * CHAR_YEXT),
                VideoFRAME[0].maprect.xext + (3 * CHAR_XEXT),
                VideoFRAME[0].maprect.yext + (3 * CHAR_YEXT),
                pBufferSize );
dlg_draw( &VideoDIALOG_BOX );
do
{
    DLGReturnCode = dlg_run( &VideoDIALOG_BOX );
}
while( (DLGReturnCode != VideoCANCELtag)
        && (DLGReturnCode != VideoOKtag) );
vid_put_screen( VideoFRAME[0].maprect.xorg - (2 * CHAR_XEXT),
                VideoFRAME[0].maprect.yorg - (2 * CHAR_YEXT),
                VideoFRAME[0].maprect.xext + (3 * CHAR_XEXT),
                VideoFRAME[0].maprect.yext + (3 * CHAR_YEXT),
                pBufferSize );
free( pBufferSize );
if( RedrawFlag == TRUE )
    Video_Draw_Screen();
else
    Video_Draw_Coordinates();
return( DLGReturnCode );
}

```


High-Level File I/O - FILEIOHL.PDM

FILEIOHL.PDM is an example of a DeskMate application which uses the GUF resource to perform its high-level file input and output functions. The source to the FileIOHL application is included in the SAMPLES\FILEIO\HIGH directory.

```
#include "fileiohl.h" /* Application header file */
#include "filehdec.h" /* Application function declarations */
```

```
extern int dmerrno;
```

```
int main( argc, argv )
int argc;
char *argv[];
{
```

Allocates memory for the file so that it may be loaded.

```
/* ask the system for at most 60K of memory */
for( pFileioBufferPointer = 0, FileioBufferSize = 0xF000;
    pFileioBufferPointer == 0; FileioBufferSize += 0x100 )
    pFileioBufferPointer = (unsigned char *)malloc( FileioBufferSize );

/* add in the extra 100 hex bytes that was taken out the last time around */
FileioBufferSize += 0x100;
```

Initialize Datafile Pointers so we can use the High Level `fil_menu_*` calls. These calls require a pointer to a datafile structure.

```
FileioDATAFILE.pStart = pFileioBufferPointer;
FileioDATAFILE.pTop = pFileioBufferPointer + FileioBufferSize;

/* Check to see if a filename was passed */
/* to this program on the command line */
if( argc > 1 )
{
```

This next section checks the command line for a filename to load, copies the command line filename into the Datafile structure, and validates the syntax of the filename, then attempts to open the file passed on the command line.

```
/* put the command line arg into the programs datafile struct */
/* so that the filename will be displayed on the title line */
/* and the data file can be opened and loaded */
strcpy( FileioDATAFILE.pFilename, argv[1] );

/* verify the file name passed (Run command may have been used) */
if ( valid_filename( FileioDATAFILE.pFilename,
                    FileioDATAFILE.pExtension ) == FALSE )
{
    /* A message appears when the filename is invalid */
    /* the filename was invalid so clear the file name */
    /* in the structure so it will display "(Untitled)" */
    strcpy( FileioDATAFILE.pFilename, "" );
}
else
{
    /* open and load the validated file name file */
    FileioDATAFILE.FileSize = fil_menu_open( &FileioDATAFILE,
                                            OPEN_NO_DIALOG );
}
}
else
{
```

If the file did not load successfully then the application should disable the save menu option of the file menu. For example:

```
/* disable save menu item because there */
/* is no file currently in memory */
FileMenuItems[SAVE_INDEX].bEnabled = DISABLED;
}
```


The `FileioHL_Draw_Screen` routine draws the main application screen and enters the main event loop, this is the same functionality as the `Video` and `Welcome` applications.

```

/* Draw the main screen */
FileioHL_Draw_Screen();

/* initialize the do while control flag */
Done = FALSE;

/* Process the user inputs and actions */
do
{
    /* read an event from an input device */
    event_read( &Event );

    switch( Event.msg )
    {
        case EVENT_COMMAND :

            /* check to see if an item was selected from the menu bar */
            /* process menu item that was selected */

            switch( Event.param )
            {

```

Processing the "New" command takes the application to a no file loaded state. The application should clear memory, and have no file loaded. Once again `DISABLE` the save menu option on the `FILE` menu. `fil_menu_new` initializes the structure filename to a null string so that when `ttl_put_data_name` is called "(Untitled)" will be displayed. The screen is then redrawn.

```

case FILE_NEW_ID:
    /* the user wants to clear out all previously */
    /* entered data, and go to a default new state */
    FileReturnCode = fil_menu_new( &FileioDATAFILE );
    if( FileReturnCode == TRUE )
    {
        /* everything went well and we need to */
        /* go to a default/no file loaded state */

        /* disable save menu item because there */
        /* is no file currently in memory */
        FileMenuItems[SAVE_INDEX].bEnabled = DISABLED;

        /* Display file name on the title line */
        /* Sending a pointer to a null string */
        /* will display "Untitled" */
        ttl_put_data_name( FileioDATAFILE.pFilename );
    }
    if( FileReturnCode == DM_ERROR && dmerrno ==
        DMERR_NONDESTRUCTIVE_ABORT )
    {
        /* the file could not be saved but the current */
        /* data is not destroyed so just redisplay the */
        /* current file status information */
    }
    FileioHL_Draw_Status_Info();
    break;

```

The open section prompts the user with the standard open dialog box for them to select a file. (The filename is syntatically correct since `valid_filename` has been called). If the return from `fil_menu_open` is `DM_ERROR` then the application should check `dmerrno` for `DMERR_INVALID_FILE_TYPE` to make sure the file being loaded belongs with that application. Some applications may accept more than one file type (like this one). If the `FileType` element

is NULL then the FileType will not be checked (no verify). Then the application should attempt to reopen the file (with no dialog box, since the filename is already correct). On a successful load, the save menu option in the file menu is enabled. The filename is displayed via the ttl_put_data_name call.

```

case FILE_OPEN_ID:
/* the user wants to open a new file */
/* and load it into memory */
TempFileSize = fil_menu_open( &FileioDATAFILE,
                                OPEN_WITH_DIALOG );
if( TempFileSize == DM_ERROR )
{
    if( dmerrno == DMERR_INVALID_FILE_TYPE )
    {
        /* the file type is not an ASCII Text file */
        strcpy( FileioDATAFILE.pFilename,
                FileioDATAFILE.pTmpfil );

        /* do not do any checking on the file type */
        FileioDATAFILE.FileType = CSR_NULL;

        /* Force the open with no dialog and load it */
        TempFileSize = fil_menu_open( &FileioDATAFILE,
                                        OPEN_NO_DIALOG );
        FileioDATAFILE.pEnd = pFileioBufferPointer
                               + TempFileSize;

        /* set the FileSize in the datafile structure */
        FileioDATAFILE.FileSize = TempFileSize;

        /* save the current FileSize for later use */
        /* see DMERR_NONDESTRUCTIVE_ABORT in open */
        OldFileSize = TempFileSize;

        /* a filename exists so enable save & saveas */
        FileMenuItems[SAVE_INDEX].bEnabled = ENABLED;

        /* Display the application's new data */
        /* file name on the title line */
        ttl_put_data_name( FileioDATAFILE.pFilename );
    }
}

```

The new file could not be loaded. So save the old filesize.

```

if( dmerrno == DMERR_NONDESTRUCTIVE_ABORT )
{
    /* the previous file is intact but the */
    /* new file could not be opened/loaded */

    /* reset the file size to the old file */
    FileioDATAFILE.FileSize = OldFileSize;

    /* redraw the current file info*/
}

```

When the error DMERR_DESTRUCTIVE_ABORT is returned it means that when the new file was trying to be loaded, it corrupted the one that was currently in memory so the application has nothing to go back to. Therefore the application should go to a default or no file loaded state. Display file name on the title line. Sending a pointer to a null string will display "Untitled".

```

if( dmerrno == DMERR_DESTRUCTIVE_ABORT )
{
    ttl_put_data_name( FileioDATAFILE.pFilename );

    /* set the FileSize in the datafile structure */
    /* it should be equal to DM_ERROR */
    FileioDATAFILE.FileSize = TempFileSize;
}

```


Redraw the screen.

```
/* erase the open dialog box */
vid_move_cursor( 0, op_row - (2*CHAR_YEXT) );
vid_clear_to_bot();

/* redraw the current file status information */
FileioHL_Draw_Status_Info();

break;
}
```

Successful file open.

```
/* Display the application's new data */
/* file name on the title line */
ttl_put_data_name( FileioDATAFILE.pFilename );

/* there is now a filename so enable save & saveas */
FileMenuItems[SAVE_INDEX].bEnabled = ENABLED;

/* erase the open dialog box */
vid_move_cursor( 0, op_row - (2*CHAR_YEXT) );
vid_clear_to_bot();
```

The application must update the Datafile structures with the size of the file (returned from open) to the pointer pEnd to the end of the data

```
/* set the end to equal the file size */
FileioDATAFILE.pEnd = pFileioBufferPointer
                    + TempFileSize;

/* set the FileSize in the datafile structure */
FileioDATAFILE.FileSize = TempFileSize;

/* save the current FileSize for later use */
/* see DMERR DESTRUCTIVE ABORT in open above */
OldFileSize = TempFileSize;

/* redraw the current file status information */
FileioHL_Draw_Status_Info();
break;
```

Saves the applications data. It is not critical to this application that the save function be successful, your specific application would probably go through more error checking than does this simple example. Once the file was saved, the application should redraw the main screen.

```
case FILE_SAVE_ID:
/* the user wants to save the current file */
if( fil_menu_save( &FileioDATAFILE ) == TRUE )
/* The data was successfully saved */

/* redraw the current file status information */
FileioHL_Draw_Status_Info();
break;
```

The Save As function should perform the same functionality as the save function as described above. The only difference is that Save As should prompt the user to enter a new filename for the data to be save into. After prompting the user the application saves the data in the new filename, displays the new filename on the title portion of the screen, and redraws the main or default screen.

```
case FILE_SAVE_AS_ID:
/* the user wants to save the current */
/* file with a different name */
if( fil_menu_saveas( &FileioDATAFILE ) == TRUE )
/* The data was successfully saved */

/* clear the save as dialog box from the screen */
vid_move_cursor( 0, sa_row - (2*CHAR_YEXT) );
vid_clear_to_bot();
```



```

/* Display the application's new data */
/* file name on the title line */
ttl_put_data_name( FileioDATAFILE.pFilename );

/* redraw the current file status information */
FileioHL_Draw_Status_Info();
break;

```

The user requested to exit the application. This application does not have any cleanup work to do. If your application does it would do it here. The `fil_menu_quit` routine prompts the user to save any data changed since the last change. If the user answers CANCEL then `fil_menu_quit` returns FALSE to the application so that it will not exit.

```

case FILE_EXIT_ID:
/* setup for exit */
if( fil_menu_quit( &FileioDATAFILE ) == TRUE )
    Done = TRUE;
else
    FileioHL_Draw_Status_Info();
break;

```

The `fil_menu_run` routines functions in the same way as `fil_menu_quit`, with the exception that the user may press CANCEL at the Run File dialog box, which would cancel the run/quit function. If the user does not cancel either option then the application must setup to exit, as described above. `fil_menu_run` will then run another application instead of returning to the Desktop or to DOS.

```

case FILE_RUN_ID:
/* setup for exit then call fil menu run */
if( fil_menu_run( &FileioDATAFILE ) == TRUE )
    Done = TRUE;
else
{
    /* the user cancelled the run action or there */
    /* was an error when trying to save the file */
    /* Regardless the run dialog box needs to be */
    /* cleared from the screen */
    vid_move_cursor( 0, FIO_RUN_YORG - (2*CHAR_YEXT) );
    vid_clear_to_bot();
    FileioHL_Draw_Status_Info();
}
break;

```

The `about_versions` call is the standard DeskMate way to display version information to the user. This routine is also used in the Video and Welcome applications.

```

case FILE_ABOUT_ID:
/* make a DeskMate library call to */
/* display an ABOUT... dialog box */
about_versions( &FileioHLAPPL_VERSION );

/* clear the about dialog box from the screen */
vid_move_cursor( 0, 3 * CHAR_YEXT );
vid_clear_to_bot();

/* display the File Status info back on the screen */
FileioHL_Draw_Status_Info();
break;

```

The modified option is an application specific function that allows the user to change the status of the Modified flag in the Datafile structure. This is not a normal application function, it is done here so that the programmer can see the messages that are displayed to the user once a "real" file is modified.

```

case OPTION_MODIFIED_ID:
/* this option allows the user to change */
/* the status of the modified flag */
if( FileioDATAFILE.Modified == TRUE )
{
    /* mark the file as "un-modified" */
    FileioDATAFILE.Modified = FALSE;
}

```



```

        /* un-check the menu item */
        OptionMenuItems[MODIFIED_INDEX].bChecked
        = MB_UNCHECKED;
    }
    else
    {
        /* mark the file as "modified" */
        FileioDATAFILE.Modified = TRUE;

        /* check the menu item */
        OptionMenuItems[MODIFIED_INDEX].bChecked
        = MB_CHECKED;
    }

    /* display the File Status info back on the screen */
    FileioHL_Draw_Status_Info();
    break;

```

The redraw screen option the programmer the ability to redraw the entire screen. This option is not usually available to the users, but it's functionality is all but required for most DeskMate applications.

```

        case OPTION_REDRAW_ID:
            /* this option allows the user to */
            /* redraw the screen at any time */

            /* display the File Status info back on the screen */
            FileioHL_Draw_Screen();
            break;

    } /* end of switch on type of application event */
    break;

    :
    :
}
/* check to see if "EXIT" or "RUN" menu item has been selected */
while( Done != TRUE );

/* give back the memory previously malloc'd */

```

This application needs to release the memory previously allocated. This could have been done in the exit and run functions above, but it was done in this application here to conserve code. After the memory has been released, the bindings to the resources are released, and the application exits.

```

    free( pFileioBufferPointer );

    /* inform the loaded resources that the application is exiting */
    guf_bind_end();
    csr_end();
    exit(0);

} /* end of Fileio main module */

```

The FileioHL_Draw_Screen function displays the application name and the name of the datafile, and draws the file status information. This is very similar to the other two sample applications discussed earlier.

```

void FileioHL_Draw_Screen()
{
    /* Display the application's name on the title line */
    ttl_put_app_name( "FileioHL" );

    /* Display the application's data file name on the title line */
    /* Sending a pointer to a null string will display "Untitled" */
    ttl_put_data_name( FileioDATAFILE.pFilename );

    FileioHL_Draw_Status_Info();
}

```

FileioHL_Draw_Status_Info displays the information in the Datafile structure.

Low-Level File I/O - FILEIOLL.PDM

FILEIOLL.PDM is an example of a DeskMate application which uses the GUF resource to perform its Low-level file input and output functions. The source to the FileIOLL application is included in the SAMPLES\FILEIO\LOW directory.

```
#include "fileioll.h" /* Application header file */
#include "fileldec.h" /* Application function declarations */
```

To keep the code simpler and clearer this application uses the Datafile structure, even though the low level routines do not require it.

```
int main( argc, argv )
int argc;
char *argv[];
{
```

The buffer pointers are initialized ,the command line is checked, the filename is copied into the Datafile structure the same as they were in the high level example discussed earlier. Then the file is opened via an application defined routine FileioLL_Load_DataFile which functions similar to the DeskMate function fil_menu_open. The main screen is drawn and the application enters the main event loop.

```
/* initialize pStart and pEnd in DATAFILE structure */
FileioDATAFILE.pStart = FileioDATAFILE.pEnd = pFileioBufferPointer;

/* initialize pTop to point to the last available byte of memory */
FileioDATAFILE.pTop = (pFileioBufferPointer + FileioBufferSize) - 1;

/* Check to see if a filename was passed */
/* to this program on the command line */
if( argc > 1 )
{
    /* put the command line arg into the programs datafile struct */
    /* so that the filename will be displayed on the title line */
    /* and the data file can be opened and loaded */
    strcpy( FileioDATAFILE.pFilename, argv[1] );

    LDReturnCode = FileioLL_Load_DataFile( OPEN_NO_DIALOG );
    if( LDReturnCode == FALSE )
    {
        /* the file could not be opened */
        /* disable save menu item because there */
        /* is no file currently in memory */
        FileMenuItems[SAVE_INDEX].bEnabled = DISABLED;
    }
    else
    {
        /* the file open was successful so enable */
        /* the save & save as menu items */
        FileMenuItems[SAVE_INDEX].bEnabled = ENABLED;
    }
    else
    {
        /* disable save menu item because there */
        /* is no file currently in memory */
        FileMenuItems[SAVE_INDEX].bEnabled = DISABLED;
    }

    /* Draw the main screen */
    FileioLL_Draw_Screen();

    /* initialize the do while control flag */
    Done = FALSE;

    /* Process the user inputs and actions */
    do
    {
        /* read an event from an input device */
        event_read( &Event );
```



```

switch( Event.msg )
{
    case EVENT_COMMAND :
        /* check to see if an item was selected from the menu bar */
        /* process menu item that was selected */

        switch( Event.param )
        {

```

The new operation checks the status of the modified flag in the Datafile structure and prompts the user to save changes (if necessary). Once the file is successfully saved, the application initializes all variables to go to a new untitled state. The filename is displayed and the File I/O main screen is redrawn.

```

    case FILE_NEW_ID:
        /* the user wants to clear out all previously */
        /* entered data, and go to a default new state */

        /* assume the user will not cancel this operation */
        CancelFlag = FALSE;

        /* check the DataFile modified flag */
        if( FileioDATAFILE.Modified == TRUE )
        {
            SCReturnCode = msgbox SaveChanges();
            if( SCReturnCode == MSG_YES )
            {
                /* the user wants to save the current file */
                FileioLL_Save_DataFile(
                    FileioDATAFILE.pFilename );
                CancelFlag = FALSE;
            }

            if( SCReturnCode == MSG_NO )
            {
                /* the user does not want to save the */
                /* current file, but wants to continue */
                /* the new operation */
                CancelFlag = FALSE;
            }

            if( SCReturnCode == MSG_CANCEL )
            {
                /* the user wants to cancel the new operation */
                CancelFlag = TRUE;
            }
        }
        if( CancelFlag == TRUE )
        {
            FileioLL_Draw_Status_Info();
            break;
        }

        /* reinitialize variables to a "new" state */
        FileioLL_New_State();

        /* Display file name on the title line */
        /* Sending a pointer to a null string */
        /* will display "Untitled" */
        ttl_put_data_name( FileioDATAFILE.pFilename );

        FileioLL_Draw_Status_Info();
        break;

```

FileioLL_Load_DataFile is an application written subroutine which provides the same functionality to the application as the DeskMate function `fil_menu_open`. FileioLL_Load_Datafile returns FALSE if the open was unsuccessful, so that the application may handle any errors which may occur, for this example the save option is

DISABLED on load datafile failure. If everything is successful the new filename should be displayed on the title line, the open dialog box should be removed from the screen and the main default screen should be drawn.

```

case FILE_OPEN_ID:
/* the user wants to open a new file */
/* and load it into memory */
LDReturnCode = FileioLL_Load_DataFile(
                                OPEN_WITH_DIALOG );
if( LDReturnCode == FALSE )
{
    /* the load was unsuccessful */
    /* there is still no filename so disable save */
    FileMenuItems[SAVE_INDEX].bEnabled = DISABLED;
}
else
{
    /* the load was successful */
    /* there is now a filename so enable save */
    FileMenuItems[SAVE_INDEX].bEnabled = ENABLED;
}

/* Display the application's new data */
/* file name on the title line */
ttl_put_data_name( FileioDATAFILE.pFilename );

/* erase the open dialog box */
vid_move_cursor( 0, op_row - (2*CHAR_YEXT) );
vid_clear_to_bot();

/* redraw the current file status information */
FileioLL_Draw_Status_Info();
break;

```

FileioLL_Save_DataFile is an application written subroutine which provides the same functionality to the application as the DeskMate **fil_menu_save** call. Once the file is successfully save the main default screen should be redrawn.

```

case FILE_SAVE_ID:
/* the user wants to save the current file */
/* a filename should already exist, since this */
/* option is grayed unless there is a valid file name */
SReturnCode = FileioLL_Save_DataFile(
                                FileioDATAFILE.pFilename );
if( SReturnCode == FALSE )
{
    /* the save was unsuccessful */
}
else
{
    /* The data was successfully saved */
}

/* redraw the current file status information */
FileioLL_Draw_Status_Info();
break;

```

digbox_SaveAs is a DeskMate function which displays a dialog box for the user to enter the new filename, so the data in memory can be saved to that new file. Once again the application defined routine FileioLL_Save_Datafile is called and functions the same as it did for the save function described above. After the save is successful, the application must remove the Save As dialog box from the screen, display filename on title line and redraw the main default screen.


```

case FILE_SAVE_AS_ID:
    /* the user wants to save the current */
    /* file with a different name */
    SReturnCode = dlgbox_SaveAs( &FileioDATAFILE,
                                SaveAsFilename );
    if( SReturnCode == TRUE )
    {
        /* it is ok to try and save the file */
        SReturnCode = FileioLL_Save_DataFile(
            SaveAsFilename );
        if( SReturnCode == FALSE )
        {
            /* the save was unsuccessful */
        }
        else
        {
            /* The data was successfully saved */
        }
    }
    else
    {
        /* the user decided not to save the file */
        /* or there was an error from dlgbox_SaveAs */
    }

    /* clear the save as dialog box from the screen */
    vid_move_cursor( 0, sa_row - (2*CHAR_YEXT) );
    vid_clear_to_bot();

    /* Display the application's new data */
    /* file name on the title line */
    ttl_put_data_name( FileioDATAFILE.pFilename );

    /* redraw the current file status informtion */
    FileioLL_Draw_Status_Info();
    break;

```

The exit function provides the same function in this application as it did in the FileioHL application described earlier. The difference being that this application must prompt the user to save changes if the data has been modified since the last save, where this was automatic in the FileioHL application. The low level application must provide much more functionality and error checking as you can see. If the user wants to save the data there must be a filename, so the application must prompt the user and process all possible combinations of error and return codes.

```

case FILE_EXIT_ID:
    /* seTup for exit */
    if( FileioDATAFILE.Modified == TRUE )
    {
        SReturnCode = msgbox_SaveChanges();
        switch ( SReturnCode )
        {
            case MSG_YES:
                /* the user wants to save current file */
                if( FileioDATAFILE.pFilename == '\0' )
                {
                    /* there is not a current filename */
                    /* prompt the user for a filename */
                    SReturnCode = dlgbox_SaveAs(
                        &FileioDATAFILE,
                        SaveAsFilename );
                    if( SReturnCode == TRUE )
                    {
                        /* try and save the file */
                        SReturnCode = FileioLL_Save_DataFile(
                            SaveAsFilename );
                        if( SReturnCode == FALSE )
                        {
                            /* the save failed */
                            break;
                        }
                    }
                }
            }
        }
    }

```



```

        else
        {
            /* the user did not specify a */
            /* name to save the file into */
            break;
        }
    }
    else
    {
        /* there is a current filename */
        SDReturnCode = FileioLL_Save_Datafile(
            SaveAsFilename );
        if( SDReturnCode == FALSE )
            /* the save failed */
            break;
    }

    /* close the currently open file */
    FileioLL_Close_File();

    Done = TRUE;

    break;

/* The user does not want to save the file */
/* but wants to continue the exit operation.*/
case MSG_NO:
    /* the user does not want to save file */
    /* close the currently open file */
    FileioLL_Close_File();

    Done = TRUE;
    break;

case MSG_CANCEL:
    /* the user wants to cancel the new */
    default:
        break;

} /* end of switch on SReturnCode */
} /* the modified flag is FALSE */

/* close the currently open file */
FileioLL_Close_File();

/* set flag so the application will exit. */
Done = TRUE;

break;

.
.
} /* end of main module */

```


FileioLL_Load_DataFile is an application written subroutine that provides the save functionality to the application as **fil_menu_open**. It checks to see if the user needs to be prompted to get a filename if so **dlgbox_Generic** (a DeskMate generic open dialog box function) is called. The filename is validated for syntax errors, if bad this routines goes to a new state and exits FALSE to the calling routine. Then the file is opened. If the application specified a FileType the specific bytes in the file must be checked. The first four are critical. The first byte contains the FileType, the next three are the extension of the file. A comparison is done on the all of these bytes, if they do not match, the application goes to a "new" state and return FALSE to the calling function. The next eighteen bytes are assumed to be the page setup information, they are read in and DeskMate's internal structures are initialized. After the remainder of the file is read in the top of memory pointer pEnd is updated, this is the same as the high level example.

```
int FileioLL_Load_DataFile( bVerbose )
int bVerbose;
{
    int Successful;
    char TempBuf[4];
    long LSReturnCode;
    int ReturnCode;
    int DGReturnCode;
    unsigned int NumBytesRead;
    unsigned int VFReturnCode;
    unsigned int FAEReturnCode;
    unsigned int FilReturnCode;

    /* have a positive outlook, assume we will be successful */
    Successful = TRUE;

    /* determine if the dialog box needs to be displayed */
    if( bVerbose == OPEN_WITH_DIALOG )
    {
        /* run the open dialog box to get the data file name */
        DGReturnCode = dlgbox_Generic( FileioDATAFILE.pExtension,
                                     FileioDATAFILE.pFilename, FIO_OPEN );
        if( DGReturnCode == FALSE )
        {
            /* the data file name was not received */
            FileioLL_New_State();
            Successful = FALSE;
            return( FALSE );
        }
    }
    else
    {
        /* the following valid filename & file already exists */
        /* calls are made by the dlgbox_Generic call so we do */
        /* not have to execute them for the case above */

        /* validate the file name passed (Run command may have been used) */
        VFReturnCode = valid_filename( FileioDATAFILE.pFilename,
                                     FileioDATAFILE.pExtension );
        if( VFReturnCode == FALSE )
        {
            /* A message appears when the filename is invalid */
            /* The file name was invalid so clear the file name */
            /* go to a "new" state */
            FileioLL_New_State();
            Successful = FALSE;
            return( FALSE );
        }
    }
}
```



```

/* check to make sure the file already exists on disk */
FAEReturnCode = file_already_exists( FileioDATAFILE.pFilename );
if( FAEReturnCode == FALSE )
{
    /* The file does not already exist, so */
    /* display a message box to tell the user */
    Msg.pMessage = "The file does not already exist";
    Msg.pString = "File error";
    Msg.btn_combo = MSG_COMBO_OK;
    msg_run( &Msg );
    /* go to a "new" state */
    FileioLL New State();
    Successful = FALSE;
    return( FALSE );
}

/* Open and load the validated file name file */
FilReturnCode = fil_open( FileioDATAFILE.pFilename,
                          OPEN_FOR_UPDATE | EXCLUSIVE );
if( FilReturnCode == DM_ERROR )
{
    /* The open failed */
    fil_open_error_msg();
    /* go to a "new" state */
    FileioLL New State();
    Successful = FALSE;
    return( FALSE );
}
else
{
    /* The open was successful so before the FileHandle is set */
    /* to the new FileHandle (via FilReturnCode) the old file */
    /* needs to be closed to avoid having too many files open */
    /* at a single time. In this manner the most files open at */
    /* one time will be two. */
    FileioLL Close File();
    FileioDATAFILE.FileHandle = FilReturnCode;
}

/* Get the file size of the open file so we can check to see */
/* if we have enough room to load the file.*/
/* get the size of the file by seeking to the end */
LSReturnCode = fil_lseek( FileioDATAFILE.FileHandle, 0L, 2 );
if( LSReturnCode == -1L )
{
    /* Tell user about any error via a DeskMate call.*/
    fil_lseek_error_msg();
    /* go to a "new" state */
    FileioLL New State();
    Successful = FALSE;
    return( FALSE );
}
else
    FileioDATAFILE.FileSize = LSReturnCode;

/* reposition the file pointer to the beginning of file before read */
LSReturnCode = fil_lseek( FileioDATAFILE.FileHandle, 0L, 0 );
if( LSReturnCode == -1L )
{
    fil_lseek_error_msg();
    Successful = FALSE;
    /* go to a "new" state */
    FileioLL New State();
    return( FALSE );
}
/* the check for the file size will come after checking */
/* for, and/or reading in file header information */

/* Check for DeskMate file header information */
/* Assume ascii file */
if( FileioDATAFILE.FileType != 0 )
{
    /* Initialize temp buffer to -1's */
    TempBuf[0] = TempBuf[1] = TempBuf[2] = TempBuf[3] = 0xFF;
}

```



```

/* Read the first byte of the file which */
/* is, in this case, the FileType */
FilReturnCode = fil_read( FileioDATAFILE.FileHandle, TempBuf, 1 );
if( FilReturnCode == DM_ERROR )
{
    /* the file could not be read */
    fil_read_error_msg();
    /* go to a "new" state */
    FileioLL New State();
    Successful = FALSE;
    return( FALSE );
}

/* Verify FileType */
if( TempBuf[0] != FileioDATAFILE.FileType )
{
    /* FileType is not correct */
    /* go to a "new" state */
    FileioLL New State();
    Successful = FALSE;
    return( FALSE );
}

/* Read in the applications extension, the next */
/* three bytes of the file header information */
FilReturnCode = fil_read( FileioDATAFILE.FileHandle, TempBuf, 3 );
if( FilReturnCode == DM_ERROR )
{
    /* file could not be read */
    fil_read_error_msg();
    /* go to a "new" state */
    FileioLL New State();
    Successful = FALSE;
    return( FALSE );
}

/* Compare what was returned from the read */
/* above to what is in the DATAFILE structure */
ReturnCode = strncmp( FileioDATAFILE.pExtension, TempBuf, 3 );
if( ReturnCode != 0 )
{
    /* go to a "new" state */
    FileioLL New State();
    Successful = FALSE;
    return( FALSE );
}

/* Get the page setup information from DeskMate */
ptd_get_page( &PGSetup, &PGMode );

/* The next 18 bytes are printer setup information */
/* Read them in. */
FilReturnCode = fil_read( FileioDATAFILE.FileHandle,
                          (char *)&PGSetup, 18 );
if( FilReturnCode != 18 )
{
    /* file could not be read */
    fil_read_error_msg();
    /* go to a "new" state */
    FileioLL New State();
    Successful = FALSE;
    return( FALSE );
}

/* Initilize the page setup in DeskMate to be */
/* the same as the files page setup information */
ptd_set_page( &PGSetup, &PGMode );

/* Adjust the size of the file for the file too large calculation */
FileioDATAFILE.FileSize -= FILE_HEADER_LENGTH;
}

```



```

/* Read in the rest of the file, */
/* or jump to here if the file type was NULL. */
/* Check to see if the size of the file will fit into memory */
if( (unsigned)FileioDATAFILE.FileSize > FileioBufferSize )
{
    /* The file will not fit into the allocated buffer */
    /* so display a message box to the user */
    Msg.pMessage = "The file is too large to fit into allocated memory";
    Msg.pString = "File error";
    Msg.btn combo = MSG_COMBO_OK;
    msg_run( &Msg );
    /* go to a "new" state */
    FileioLL New State();
    Successful = FALSE;
    return( FALSE );
}

/* Read file into allocated buffer */
NumBytesRead = fil_read( FileioDATAFILE.FileHandle, FileioDATAFILE.pStart,
                        FileioBufferSize );
if( NumBytesRead == DM_ERROR )
{
    /* The file could not be read */
    fil_read_error_msg();
    /* go to a "new" state */
    FileioLL New State();
    Successful = FALSE;
    return( FALSE );
}
else
    /* Make pEnd point to the last byte of data */
    FileioDATAFILE.pEnd = (FileioDATAFILE.pStart + NumBytesRead) - 1;

if( Successful )
    return( TRUE );
else
    return( FALSE );
}

```

FileioLL_Save_DataFile closes the file currently open, opens the new file, and saves the data in memory to the new file. pNewFilename is a pointer to the filename to be written. It may be the same as the current file, as in the case of save, or it may be a completely new filename as in the case of saveas. A check is made for DeskMate header information so that it may be skipped on the writing of data from memory. DeskMate's page setup and page mode information for this application is read in and written to the file. After the file is successfully written the Datafile structure is updated.

```

int FileioLL_Save_DataFile( pNewFilename )
char *pNewFilename;
{
    int Successful;
    long LSReturnCode;
    unsigned int FilReturnCode;
    unsigned int NumBytesWritten;
    unsigned int MemorySize;

    /* assume this function will succeed */
    Successful = TRUE;

    /* close the current file */
    FileioLL_Close_File();

    /* initialize the FileioDATAFILE structure with the new name */
    FileioDATAFILE.pFilename = pNewFilename;
}

```

Open the new validated file name file with create, the file is opened with create in case the file has gotten smaller, although it should never happen in this example, it is necessary to show what procedure an application should actually have to do. Open with create to capture possible lost disk space on a smaller file.


```

FilReturnCode = fil_create( FileioDATAFILE.pFilename );
if( FilReturnCode == DM_ERROR )
{
    /* the create failed */
    fil_create_error_msg();
    /* go to a "new" state */
    FileioLL New State();
    Successful = FALSE;
    return( FALSE );
}
else
{
    /* initialize the file handle so it may be */
    /* closed before it is reopened below */
    FileioDATAFILE.FileHandle = FilReturnCode;
    FileioLL_Close_File();
}

/* the filename validation was done by dlgbox SaveAs */
/* or in the case of save, in FileioLL Load DataFile */
FilReturnCode = fil_open( FileioDATAFILE.pFilename,
                           OPEN_FOR_UPDATE | EXCLUSIVE );
if( FilReturnCode == DM_ERROR )
{
    /* the open failed */
    fil_open_error_msg();
    Successful = FALSE;
    return( FALSE );
}
else
    FileioDATAFILE.FileHandle = FilReturnCode;

/* Check for DeskMate file header information */
/* Assume ascii file */
if( FileioDATAFILE.FileType != 0 )
{
    /* the header information must be written out */

    /* write the FileType */
    FilReturnCode = fil_write( FileioDATAFILE.FileHandle,
                              &(FileioDATAFILE.FileType), 1 );
    if( FilReturnCode != 1 )
    {
        /* the write failed */
        fil_write_error_msg();
        /* go to a "new" state */
        FileioLL New State();
        Successful = FALSE;
        return( FALSE );
    }

    /* write the Extension */
    FilReturnCode = fil_write( FileioDATAFILE.FileHandle,
                              FileioDATAFILE.pExtension, 3 );
    if( FilReturnCode != 3 )
    {
        /* the write failed */
        fil_write_error_msg();
        /* go to a "new" state */
        FileioLL New State();
        Successful = FALSE;
        return( FALSE );
    }

    /* get the page setup information from DeskMate */
    ptd_get_page( &PGSetup, &PGMode );

    /* write the Page Setup information */
    FilReturnCode = fil_write( FileioDATAFILE.FileHandle,
                              (char *)&PGSetup, 18 );
    if( FilReturnCode != 18 )
    {
        /* the write failed */
        fil_write_error_msg();
        /* go to a "new" state */
        FileioLL New State();
        Successful = FALSE;
        return( FALSE );
    }
}
}

```



```

/* calculate the size of the data in memory */
MemorySize = (unsigned) ( FileioDATAFILE.pEnd -
                           FileioDATAFILE.pStart ) + 1 );

/* write the data in memory to a disk file */
NumBytesWritten = fil_write( FileioDATAFILE.FileHandle,
                             FileioDATAFILE.pStart, MemorySize );
if( NumBytesWritten == DM_ERROR )
{
    /* the write failed */
    fil_write_error_msg();
    /* go to a "new" state */
    FileioLL_New_State();
    Successful = FALSE;
    return( FALSE );
}
else
    FileioDATAFILE.FileSize = NumBytesWritten;
}

```

FileioLL_Close_File closes the currently open file, if **DM_ERROR** is in the **FileHandle** element it indicates there is no file currently open.

```

void FileioLL_Close_File()
{
    int FilReturnCode; /* used for the fil_close call */

    /* check to see if the file handle is valid */
    if( FileioDATAFILE.FileHandle != DM_ERROR )
    {
        /* close the file */
        FilReturnCode = fil_close( FileioDATAFILE.FileHandle );
        if( FilReturnCode == DM_ERROR )
            /* I said close the file! */
            fil_force_close( FileioDATAFILE.FileHandle );
        FileioDATAFILE.FileHandle = DM_ERROR;
    }
}

```

FileioLL_New_State takes the application to a "New" state. It first closes the file, sets the Datafile structure filename element to a NULL, initializes the modified flag in the Datafile structure to FALSE, sets the Datafile structure end and start pointers to be equal, zeros the file size, and disables the save menu option.

```

void FileioLL_New_State()
{
    /* close the existing file */
    FileioLL_Close_File();

    /* NULL the current file name */
    *FileioDATAFILE.pFilename = '\0';

    /* set the modified flag to FALSE */
    FileioDATAFILE.Modified = FALSE;

    /* reset end pointer to equal start pointer */
    FileioDATAFILE.pEnd = FileioDATAFILE.pStart;

    /* reset FileSize to equal Zero */
    FileioDATAFILE.FileSize = CSR_NULL;

    /* disable save menu item because there */
    /* is no file currently in memory */
    FileMenuItems[SAVE_INDEX].bEnabled = DISABLED;
}

```


Database File I/O - DBCARS.PDM

DBCARS.PDM is an example of a DeskMate application which uses the Database resource to perform its file input and output functions. The DBCars application is included on the SAMPLES\DATABASE directory.

```
#include "dmdb.h"      /* Database Resource header file */
#include "dbcars.h"     /* Application data header file */
#include "dbdecs.h"     /* Application function prototypes header file */

main(argc, argv)
int    argc;
char   *argv[];
{
```

This application uses a database file structure which is used to store the file, table, and record information for the current file.

```
DB_DATAFILE DB_Datafile; /* Database Datafile structure */
DB_DATAFILE *pDB_Datafile; /* Pointer to the Database Datafile structure */
```

This application binds to the entire database, since it creates files and reads and updates data in the files. For more information about binding to specific database resources, see the Desk and Database Manager sections of the DeskMate Technical Reference.

```
if ( db_bind_init() == CSR_ERROR )
{
    /* failure to bind to the Database resource */
    guf_bind_end();
    csr_end();
    exit(1);
}

/* Assume untitled state */
DBCars_SetUntitled( pDB_Datafile );
```

First we check the arguments to determine if a file name was passed to the application. If an argument was passed, then the file name is validated and its full path name is generated. The database expects full path names for its data file names. The file is then opened.

```
/* Check for a file argument */
if ( argc > 1 )
{
    /* First check for a valid file name, call issues error message */
    if ( valid_filename( argv[1], "DBF" ) == TRUE )
    {
        /* Expand the name, database requires full path name */
        Path_Expand( argv[1], pDB_Datafile->Filename );

        /* Open the file passed without displaying the dialog box */
        DBCars_OpenFile( pDB_Datafile, NO_PROMPT );
    }
} /* had arguments passed on the parameter line */

/* Draw the main screen */
DBCars_DrawScreen( pDB_Datafile );
```

When we have a file opened, the first record in the file is displayed. When in the "Untitled" state, a blank screen is displayed.

```
if ( pDB_Datafile->hFile >= 0 )
{
    /* We have file, display the first record in the file */
    ModelMENUITEM[0].bEnabled = ENABLED; /* Add */
    CarsMENUBAR.bRedraw = MB_NO_REDRAW;
    mb_draw(&CarsMENUBAR);
    DBCars_ShowModel( pDB_Datafile, FIRST_RECORD );
}
```



```

/* initialize the do while control flag */
Done = FALSE;

```

This application also has a standard event processing loop. The `fil_menu_*` calls used in the File I/O examples are not used here, database calls, `db_mgr`, are instead used to do the File I/O functions.

```

/* Process the user inputs and actions */
do
{
    /* read an event from an input device */
    event_read(&Event);

    switch( Event.msg )
    {
        case EVENT_COMMAND :
            /* check to see if an item was selected from the menu bar */
            /* process menu item that was selected */
            switch( Event.param )
            {
                case FILE_NEW_ID:
                    /* the user wants to clear out all previously */
                    /* entered data, and go to a default new state */
                    DBCars_NewFile( pDB_Datafile );

                    if ( pDB_Datafile->hFile >= DB_OK )
                        /* Created file, there are no records */
                        /* Enable Add Model only, disable others */
                        DBCars_SetNoRecords();

                    /* Display the new file name, set menu */
                    DBCars_DrawScreen( pDB_Datafile );
                    break;

                case FILE_OPEN_ID:
                    /* the user wants to open a new file */
                    /* and load it into the edit field */
                    DBCars_OpenFile( pDB_Datafile, PROMPT );

                    /* Display the new file name, display the first record */
                    DBCars_DrawScreen( pDB_Datafile );
                    if ( pDB_Datafile->hFile >= 0 )
                    {
                        ModelMENUITEM[0].bEnabled = ENABLED; /* Add */
                        CarsMENUBAR.bRedraw = MB_NO_REDRAW;
                        mb_draw(&CarsMENUBAR);
                        DBCars_ShowModel( pDB_Datafile, FIRST_RECORD );
                    }
                    break;

                case FILE_EXIT_ID:
                    /* setup for exit */
                    Done = TRUE;
                    break;
            }
        }
    }
}

```

Database applications use the `dlgbox_Run` function to prompt the user for the application information for the File Run option.

```

case FILE_RUN_ID:
    /* setup for exit then call dlgbox_Run */
    if( dlgbox_Run() == TRUE )
        Done = TRUE;

    /* display the screen which will clear the */
    /* dlgbox_Run dialog box from the screen */
    DBCars_DrawScreen( pDB_Datafile );
    break;

case MODEL_ADD_ID:
    /* the user wants to add a new model to the file */
    DBCars_AddModel( pDB_Datafile );
    break;

case MODEL_MODIFY_ID:
    /* the user wants to change a models information */
    DBCars_ChangeModel( pDB_Datafile );
    break;

```



```

case MODEL DEL ID:
    /* the user wants to change a models information */
    DBCars_DeleteModel( pDB_Datafile );
    break;

case VIEW FIRST ID:
    /* the user wants to display the first record */
    DBCars_ShowModel( pDB_Datafile, FIRST_RECORD );
    break;

case VIEW PREV ID:
    /* the user wants to display the prev record */
    DBCars_ShowModel( pDB_Datafile, PREV_RECORD );
    break;

case VIEW NEXT ID:
    /* the user wants to display the next record */
    DBCars_ShowModel( pDB_Datafile, NEXT_RECORD );
    break;

case VIEW LAST ID:
    /* the user wants to display the last record */
    DBCars_ShowModel( pDB_Datafile, LAST_RECORD );
    break;

case VIEW ALL ID:
    /* the user wants to display all records */
    DBCars_Report( pDB_Datafile );
    break;

} /* end of switch on type of application event */
break;

case EVENT APPL :
    switch( Event.param )
    {
        /* check for an accessory event */
        case APPL ACCESS:
            /* run the requested accessory */
            dm_acc_run ( Event.x );
    }

```

Database applications should always verify that the correct data file disk is in the drive after running an accessory or when returning from a task switch when the user is accessing a data file on a floppy drive. The GUF functions **is_floppy** and **file_already_exists** can be used to determine if the data file is on a floppy and if the file is on the disk. If the file is not on the disk, use **dm_file_search** to prompt the user for the file.

```

        /* redraw the screen when the accessory is finished */
        DBCars_DrawScreen( pDB_Datafile );
        break;

/* check for a task switch event */
case APPL TASK SWITCH:
    /* dm_yield is the call to allow task switch to occur */
    TSReturnCode = dm_yield();

    if ( TSReturnCode == DM NOT ALLOWED )
        /* task switching not being allowed by DESK.EXE */
        break;

    if ( TSReturnCode == DM OK )
        /* The task switch has occurred so */
        /* redraw the menubar and the screen */
        DBCars_DrawScreen( pDB_Datafile );
    else
        /* the yield (task switch) failed */
        /* there is bad trouble so exit program */
        Done = TRUE;
    break;

default:
    break;
} /* end of switch on type of Command */
break;

default:
    break;

} /* end of switch on type of event */

```



```

    }
    /* check to see if "EXIT" or "RUN" menu item has been selected */
    while( Done != TRUE );

    /* Close the file */
    if ( pDB_Datafile->hFile >= 0 )
    {
        db_mgr( CLOSE_TABLE, pDB_Datafile->hTable );
        db_mgr( CLOSE_FILE, pDB_Datafile->hFile );
    }

    /* inform the loaded resources that the application is exiting */
    db_bind_end();
    guf_bind_end();
    csr_end();
    exit(0);
} /* end of Cars main module */

```

Database applications must provide their own version of the new file dialog box which is very similar to the Save as dialog box used by the `fil_menu_save_as` function. The `CREATE_FILE` database function is used to create the data file. The application must then build its file format by creating data tables in the file. Once the file structure is created, data can be added.

```

void DBCars NewFile( pDatafile )
DB_DATAFILE *pDatafile;
{
    db_table          table;
    db_columns        columns[NUM_COLS];
    db_index          index;
    int               erc;
    register int       i;
    register db_columns *pcol = &columns[0];

    /* Save off the current files name in case of problems */
    strcpy( pDatafile->TmpFilename, pDatafile->Filename );

    /* Prompt user for name for the New file */
    erc = DBCars NewFileDialog( pDatafile->Filename );
    if ( erc == FALSE )
        /* User cancelled the box, current file is okay */
        return;

    /* Create the new file for the user */
    erc = db_mgr( CREATE_FILE, pDatafile->Filename );

    if ( erc >= DB_OK )
    {
        /* Created the new file, close the current file if we have one */
        if ( pDatafile->hFile >= DB_OK )
        {
            db_mgr( CLOSE_TABLE, pDatafile->hTable );
            db_mgr( CLOSE_FILE, pDatafile->hFile );
        }
        /* save off the file handle, clear the table handle */
        pDatafile->hFile = erc;
        pDatafile->hTable = -1;
    }
    else
    {
        /* we received an error condition on the create */
        DBCars_DisplayError(erc);

        /* Reset application to previous state */
        strcpy( pDatafile->Filename, pDatafile->TmpFilename );
        return;
    }
}

```


Now that the file is created, the data tables which make up the file are created. Each row in table will correspond to a data record, each column is a field in the record. All records within a table have the same format.

```

/* the table is a standard, non-user-definable table */
table.handle = pDatafile->hFile;
table.tbl_name = sModel;
table.update_type = ADD_COLUMN;
table.n_columns = NUM_COLS;
table.n_items = NUM_COLS;
table.cols = pcol;

/* fill in the column information */
for ( i = 0; i < NUM_COLS; i++, pcol++ )
{
    pcol->col_name = colnames[i];
    pcol->col_length = collength[i];
    pcol->col_type = coltype[i];
    pcol->new_name = colnames[i];
    pcol->col_attr = 0;
    pcol->unique_flag = 0;
}

/* Add the data table to the file, returning its handle */
erc = db_mgr( CREATE_TABLE, &table );
if ( erc >= DB_OK )
    pDatafile->hTable = erc;
else
{
    /* we received an error condition on the create */
    DBCars_DisplayError(erc);

    /* Close and delete the partially created file */
    db_mgr( CLOSE_FILE, pDatafile->hFile );
    delete_file( pDatafile->Filename );

    /* Set application for the Untitled state */
    DBCars_SetUntitled(pDatafile);
    return;
}

```

Once a table is created, its sort information is defined. The default sort order is the order the records are added or modified.

```

/* Build the table index - sort by model and color */
index.table_handle = pDatafile->hTable;
index.index_name = sModelIndex;
index.pSortOrder = sortorder;

erc = db_mgr( DEFINE_INDEX, &index );
if ( erc != DB_OK )
{
    /* we received an error condition on the index create */
    DBCars_DisplayError(erc);

    /* Close and delete the partially created file */
    db_mgr( CLOSE_TABLE, pDatafile->hTable );
    db_mgr( CLOSE_FILE, pDatafile->hFile );
    delete_file( pDatafile->Filename );

    /* Set application for the Untitled state */
    DBCars_SetUntitled(pDatafile);
    return;
} /* we weren't able to create the table index */

/* Close the table which is locked from data access */
db_mgr( CLOSE_TABLE, pDatafile->hTable );

```

Now that the data file format is established, the application is ready to add data to the file.

```

/* Reopen the table for data access, go to Untitled state on an error */
DBCars_OpenTable( pDatafile );

} /* end of Cars New File module */

```


Database applications use the dialog box function **dlgbox.Generic** used by **fil_menu_open** to prompt the user for the file to open. This function returns the full path name of the file to be opened. Now the file and its data table are opened for data retrieval and updating.

```
void DBCars_OpenFile( pDatafile, bPrompt )
DB_DATAFILE *pDatafile; /* Pointer to database file structure */
int bPrompt; /* PROMPT or NO_PROMPT for file name */
{
    int    erc = TRUE;

    /* Save off the current files name in case of problems */
    strcpy( pDatafile->TmpFilename, pDatafile->Filename );

    if ( bPrompt == PROMPT )
        /* Let user choose another file */
        erc = dlgbox_Generic( "DBF", pDatafile->Filename, FIO_OPEN );

    if ( erc == FALSE )
        /* Cancelled the Open File dialog box, previous file is okay */
        return;

    /* open the named datafile */
    erc = db_mgr( OPEN_FILE, pDatafile->Filename );

    if ( erc >= DB_OK )
    {
        /* Opened the new file, close the current file if we have one */
        if ( pDatafile->hFile >= DB_OK )
        {
            /* close the table first, then the file */
            db_mgr( CLOSE_TABLE, pDatafile->hTable );
            db_mgr( CLOSE_FILE, pDatafile->hFile );
        }
        /* save off the new file's handle, clear the table handle */
        pDatafile->hFile = erc;
        pDatafile->hTable = -1;

        /* Open the data table, go to Untitled state on an error */
        DBCars_OpenTable( pDatafile );
    }
    else
        /* Could not open the new file, reset to previous state */
        strcpy( pDatafile->Filename, pDatafile->TmpFilename );
} /* end of Cars Open File module */
```

This function opens the fields data table for reading and updating.

```
void DBCars_OpenTable( pDatafile )
DB_DATAFILE *pDatafile;
{
    table_access    open_table;
    int            erc;

    /* set table information required to open the table and open it */
    open_table.file_handle = pDatafile->hFile;
    open_table.tbl_name    = sModel;
    open_table.access_level = DATA_ACCESS;
    erc = db_mgr( OPEN_TABLE, &open_table );

    if ( erc < DB_OK )
    {
        /* we received an error condition on the open */
        DBCars_DisplayError(erc);

        /* Close the file since we can't access data */
        db_mgr( CLOSE_FILE, pDatafile->hFile );

        /* Set application for the Untitled state */
        DBCars_SetUntitled( pDatafile );
    }
    /* we weren't able to open the table for data access */
    else
        pDatafile->hTable = erc;
} /* end of Cars Open Table Function */
```


This dialog box function resembles the Save as dialog box. It prompts the user for a file name and validates and expands the file name for the application.

```
int DBCars NewFileDlg( filename )
char *fiTenname;
{
    int    retval;

    /* Raise all buttons, reset the edit field, reset focus */
    NewFilePBs[0].bState = PB_UP;
    NewFilePBs[1].bState = PB_UP;

    /* Clear the data in the buffer, reset editing origin since this field */
    /* scrolls (left/right), reset the cursor and select offsets. */
    filename_buff[0] = '\0';
    NewFileEF.edit_maprect.xorg = 0;
    NewFileEF.cursor_offset = 0;
    NewFileEF.select_offset = 0;

    /* Make the edit field the current component */
    NewFileDlg.focus_index = 0;

    dlg_draw( &NewFileDlg );

    do
    {
        retval = dlg_run( &NewFileDlg );

        if ( retval == OK_tag )
        {
            if ( valid_filename( filename_buff, "DBF" ) == TRUE )
                /* Expand the file name out */
                Path_Expand( filename_buff, filename );
            else
            {
                /* Message is displayed informing user of error */
                /* Select the name and allow user to retry */
                NewFileEF.edit_maprect.xorg = 0;
                NewFileEF.cursor_offset = EF_SELECT_ALL;
                NewFileEF.select_offset = 0;
                NewFileDlg.focus_index = 0;

                /* Raise the OK pushbutton and redraw the components */
                NewFilePBs[0].bState = PB_UP;
                NewFileRedraw[0] = DLG_REDRAW;
                NewFileRedraw[1] = DLG_REDRAW;

                /* Re-run the dialog box so user can try again */
                retval = EF_tag;
            }
        } /* validate the file name entered */

    } while ( retval != OK_tag && retval != CANCEL_tag );

    if ( retval == OK_tag )
        return( TRUE );
    else
        return( FALSE );
} /* end of Cars New File Dialog Box function */
```

This sample does its record data entry through a dialog box for simplicity. Most applications allow the user to enter the information directly on the screen. See the COMPS example application for more information.

```
void DBCars AddModel( pDatafile )
DB_DATAFILE *pDatafile;
{
    int    ret;
    register int    i, j;

    /* Reset the dialog box components for the next execution of the box */
    OptionsOK.bState = OptionsCancel.bState = PB_RAISED;
    CarModelEdit.cursor_offset = 0;
    CarModelEdit.select_offset = 0;
    *CarModelEdit.pBuffer = '\0';
    OptionsDialog.focus_index = 0;
```



```

ExteriorColorGroup.selected = 0;
ExteriorColorGroup.cursor = CSR_DEFAULT;
for (i = 0; i < 6; i++)
    OptionsCheckBox[i].bState = CB_UNCHECKED;
for (i = 0; i < OptionsDialog.nCtrls; i++)
    OptionsRedrawFlag[i] = DLG_REDRAW;

dlg_draw( &OptionsDialog );

do
{
    ret = dlg_run( &OptionsDialog );
} while ( ret != ID_OK && ret != ID_CANCEL );

```

Once the user OK's the data entered, the information is transferred to the record column values buffers defined for the recd data structure.

```

if ( ret == ID_OK )
{
    /* Add the new record to the database file */
    /* fill in the record's info in the record structure */
    recd.table_handle = pDatafile->hTable;

    /* Name is ready in the record values, set the color/options states */
    /* Save selected state of exterior color radio button group */
    itoa( ExteriorColorGroup.selected, recd_vals[1].col_value, 10 );

    /* Save a 1/0 for each check box option */
    for (i = 0, j = 2; i < 6; i++, j++)
    {
        if ( OptionsCheckBox[i].bState == CB_CHECKED )
            strcpy( recd_vals[j].col_value, "1" );
        else
            strcpy( recd_vals[j].col_value, "0" );
    } /* for each check box option, save the state */
}

```

Once all the data is transferred to the recd structure, the actual ADD_ROW or add record call is made. This record's record number is returned or an error if the record could not be added.

```

/* add the record, saving the record number for the new record */
pDatafile->rec_num = db_mgr( ADD_ROW, &recd );
if ( pDatafile->rec_num < DB_OK )
    /* didn't add record, display error message */
    DBCars_DisplayError( pDatafile->rec_num );

} /* User okayed the box, add the new record */

```

The new record is displayed. Notice that the currently displayed record number is stored in the database file structure for use by the other record routines.

```

if ( pDatafile->rec_num > 0 )
    /* Display the current records information */
    DBCars_ShowModel( pDatafile, FETCH_RECORD );
else
{
    /* clear dialog box off the screen */
    vid_move_cursor( 0, 4 * CHAR_YEXT );
    vid_clear_to_bot();
}

} /* end of Cars Add Model function */

```

The same dialog box used by Add is used to change a records data. Again, the information is transferred to the data base structure and the UPDATE_ROW functions call is made to actually change the record information.

```

void DBCars_ChangeModel( pDatafile )
DB_DATAFILE *pDatafile;
{
    int          ret, err;
    register int i, j;

    /* Reset the dialog box components for execution of the box */
    CarModelEdit.cursor_offset = 0;
    CarModelEdit.select_offset = EF_SELECT_ALL;
    OptionsOK.bState = OptionsCancel.bState = PB_RAISED;
}

```



```

OptionsDialog.focus index = 0;
for ( i = 0; i < OptionsDialog.nCmps; i++ )
    OptionsRedrawFlag[i] = DLG_REDRAW;

dlg_draw( &OptionsDialog );

do
{
    ret = dlg_run( &OptionsDialog );
} while ( ret != ID_OK && ret != ID_CANCEL );

if ( ret == ID_OK )
{
    /* Name is ready in the record values, set the color/options states */
    /* Save selected state of exterior color radio button group */
    itoa( ExteriorColorGroup.selected, recd_vals[1].col_value, 10 );

    /* Save a 1/0 for each check box option */
    for ( i = 0, j = 2; i < 6; i++, j++ )
    {
        if ( OptionsCheckBox[i].bState == CB_CHECKED )
            strcpy( recd_vals[j].col_value, "1" );
        else
            strcpy( recd_vals[j].col_value, "0" );
    } /* for each check box option, save the state */

    /* Fill in the record's info in the update record structure */
    update_recd.table_handle = pDatafile->hTable;
    update_recd.rec_num = pDatafile->rec_num;
    /* update the record, we are not using data validation in this example */
    erc = db_mgr( UPDATE_ROW, &update_recd );
    if ( erc < DB_OK )
        DBCars_DisplayError( pDatafile->rec_num );

} /* update the record changes */

if ( pDatafile->rec_num > 0 )
    /* Display the current records information */
    DBCars_ShowModel( pDatafile, FETCH_RECORD );
else
{
    /* Remove the dialog box */
    vid_move_cursor( 0, 5*CHAR_YEXT );
    vid_clear_to_bot();
}

} /* end of Cars Change Model function */

```

This function deletes the current record and displays the information for the next record or informs the user that all records have been deleted.

```

void DBCars DeleteModel( pDatafile )
DB_DATAFILE *pDatafile;
{
    db_delete    delete_recd;
    int          erc;

    /* fill in information about the record to delete */
    delete_recd.table_handle = pDatafile->hTable;
    delete_recd.rec_num = pDatafile->rec_num;
    delete_recd.num_query_lines = 0;
    delete_recd.query_line_array = NULL;

    erc = db_mgr( DELETE_ROW, &delete_recd );
    if ( erc == DB_OK || erc == DB_NO_ROWS_SELECTED )
    {
        /* Remove previous records information from the screen */
        vid_move_cursor( 0, 4 * CHAR_YEXT );
        vid_clear_to_bot();

        /* Inform user and set menus for empty file state */
        DBCars_DisplayMsg("All records have been deleted.");
        DBCars_SetNoRecords();
    }
}

```



```

else
    if ( erc < DB_OK )
        DBCars_DisplayError( erc );
    else
    {
        /* we have a new current record */
        pDatafile->rec_num = erc;
        DBCars_ShowModel( pDatafile, FETCH_RECORD );
    } /* have a next record */
} /* end of Cars Delete Model function */

```

This function does single record fetching to get the data to display for a record. When a record number is not provided then a directional (first, next, etc.) fetch is made.

```

void DBCars_ShowModel( pDatafile, type_fetch )
DB_DATAFILE *pDatafile;
int type_fetch;
{
    int      erc;
    register int i, j;
    char      *pData;

    /* Setup the View Menu - make some assumptions on the result */
    if ( type_fetch == FIRST_RECORD )
    {
        ViewMENUITEM[0].bEnabled = DISABLED;    /* First */
        ViewMENUITEM[1].bEnabled = DISABLED;    /* Prev */
        ViewMENUITEM[2].bEnabled = ENABLED;      /* Next */
        ViewMENUITEM[3].bEnabled = ENABLED;      /* Last */
    } /* we know we are going to the start */
    else
    {
        if ( type_fetch == LAST_RECORD )
        {
            ViewMENUITEM[0].bEnabled = ENABLED;    /* First */
            ViewMENUITEM[1].bEnabled = ENABLED;    /* Prev */
            ViewMENUITEM[2].bEnabled = DISABLED;    /* Next */
            ViewMENUITEM[3].bEnabled = DISABLED;    /* Last */
        } /* we know we are going to the end */

        /* Will almost always have more than one record */
        ViewMENUITEM[4].bEnabled = ENABLED;        /* All */

        /* Fetch the current records data */
        get_recd.table_handle = pDatafile->hTable;
        get_recd.rec_num = pDatafile->rec_num;
        erc = db_mgr( type_fetch, &get_recd );
        pDatafile->rec_num = get_recd.rec_num;
    }
}

```

All the possible error conditions are handled and the View Menu is changed to reflect the position of this record in the file.

```

switch( erc )
{
    case DB_OK:
        /* Retrieved a record, not the first or */
        /* last so enable all paging options */
        if ( type_fetch != FIRST_RECORD && type_fetch != LAST_RECORD )
        {
            for ( i = 0; i < NUM_VIEW_MENU_ITEMS; i++ )
                ViewMENUITEM[i].bEnabled = ENABLED;
        }
        break;

    case DB_FIRST_RECORD:
        /* This is the first record */
        ViewMENUITEM[0].bEnabled = DISABLED;    /* First */
        ViewMENUITEM[1].bEnabled = DISABLED;    /* Prev */
        ViewMENUITEM[2].bEnabled = ENABLED;      /* Next */
        ViewMENUITEM[3].bEnabled = ENABLED;      /* Last */
        break;

    case DB_LAST_RECORD:
        /* This is the last record */
        ViewMENUITEM[0].bEnabled = ENABLED;    /* First */
        ViewMENUITEM[1].bEnabled = ENABLED;    /* Prev */
        ViewMENUITEM[2].bEnabled = DISABLED;    /* Next */
        ViewMENUITEM[3].bEnabled = DISABLED;    /* Last */
        break;
}

```



```

case DB_NO_RECORDS:
    /* There are not any records in the file */
    ViewMENUITEM[4].bEnabled = DISABLED; /* All */

case DB_ONLY_RECORD:
    /* This is the only record */
    ViewMENUITEM[0].bEnabled = DISABLED; /* First */
    ViewMENUITEM[1].bEnabled = DISABLED; /* Prev */
    ViewMENUITEM[2].bEnabled = DISABLED; /* Next */
    ViewMENUITEM[3].bEnabled = DISABLED; /* Last */
    break;

default:
    DBCars_DisplayError(erc);
    pDatafile->rec_num = -1;
    return;
}

ModelMENUITEM[1].bEnabled = ENABLED; /* Change */
ModelMENUITEM[2].bEnabled = ENABLED; /* Delete */

/* redraw the application menubar to change valid options */
CarsMENUBAR.bRedraw = MB_NO_REDRAW;
mb_draw(&CarsMENUBAR);

if (erc == DB_NO_RECORDS)
{
    /* Clear the screen, there isn't any information to display */
    vid_move_cursor( 0, 3 * CHAR_YEXT );
    vid_clear_to_bot();
    return;
}

```

The data is transferred to the dialog box data buffers for a possible modification of this record.

```

/* Set the current records options in the dialog box for a change */
pData = get_recd.buffer;

/* First copy over the model name, then the color selection */
strcpy( CarModelBuffer, pData );
pData += strlen(pData) + 1;
ExteriorColorGroup.selected = atoi(pData);
pData += strlen(pData) + 1;

/* Now set each of the Option check boxes */
for ( i = 0; i < 6; i++ )
{
    /* If a 1 was saved, box is checked, otherwise it is unchecked */
    OptionsCheckBox[i].bState = (atoi(pData) == 1) ?
                                CB_CHECKED : CB_UNCHECKED;
    pData += strlen(pData) + 1;
}

/* Now display the information on the screen */
vid_move_cursor( 0, 3 * CHAR_YEXT );
vid_clear_to_bot();

vid_move_cursor( 5 * CHAR_XEXT, 5 * CHAR_YEXT );
vid_put_string("Model: ");
vid_put_string( CarModelBuffer );

vid_move_cursor( 40 * CHAR_XEXT, 5 * CHAR_YEXT );
vid_put_string("Color: ");
vid_put_string( OptionsStrings[2 + ExteriorColorGroup.selected].pString );

vid_move_cursor( 25 * CHAR_XEXT, 7 * CHAR_YEXT );
vid_put_string("Options");
vid_set_line_attr( LINE_SOLID, LINE_WIDTH1, COLOR3, COLOR4 );
vid_draw_rect( 10 * CHAR_XEXT, 8 * CHAR_YEXT,
               46 * CHAR_XEXT, 15 * CHAR_YEXT, VID_NO_FILL );

```



```

for ( i = 0, j = 9; i < 6; i++ )
{
    if ( OptionsCheckBox[i].bState == CB_CHECKED )
    {
        vid_move_cursor( 12 * CHAR_XEXT, j++ * CHAR_YEXT );
        vid_put_string( OptionsStrings[9 + i].pString );
    } /* the option is set, display its string */
} /* check the next option */

} /* end of Cars Show Model function */

```

This function demonstrates the use of a multiple record query to read in the file data to display in a report format. The Fetch Records function handles the possible error conditions returned by the SETUP_QUERY and MORE_RECORDS database functions. The data is returned in a buffer which contains the record index for the records. See the Database Manager section of the DeskMate Technical Reference for detailed information about the buffer format.

```

void DBCars Report( pDatafile )
DB_DATAFILE *pDatafile;
{
    char          *pData;
    DB_INDEX_NODE *pIndex;
    register int   i, j;
    int           bLast, k;
    EVENT         event;

    /* query database for all records in the table */
    query.table_handle = pDatafile->hTable;
    bLast = DBCars FetchRecords( SETUP_QUERY, &query );
    if ( bLast == DM_ERROR )
        /* Don't display the report, had a problem */
        return;

    /* Display the report - write title line */
    vid_move_cursor( 0, 3 * CHAR_YEXT );
    vid_clear_to_bot();

    vid_move_cursor( 0, 3 * CHAR_YEXT );
    vid_set_char_attr( BOLD | UNDERLINE );
    vid_put_string( ReportStr );
    vid_set_char_attr( NORMAL );

    /* Display all records in the data file, one 20 record page at a time */
    /* Data starts at the beginning of the buffer, record index starts at */
    /* the end of the buffer (reset if more records are read). */
    /* Start the record index at the top, will be reset if more records */
    /* need to be read from the file (bLast is DM_ERROR). */
    pIndex = (DB_INDEX_NODE *) ( query.pBuffer + query.amt_memory
                                - sizeof(DB_INDEX_NODE) );
    i = 0;

    do
    {
        /* Clear the screen for a new page of records */
        vid_move_cursor( 0, 4 * CHAR_YEXT );
        vid_clear_to_bot();

        for ( j = 4; i < query.rec_cnt && j < 24; i++, j++, pIndex-- )
        {
            /* Point to this record's data */
            pData = query.pBuffer + pIndex->offset;

            vid_move_cursor( 0, j * CHAR_YEXT );
            vid_put_string( pData ); /* Model Name */
            pData += strlen( pData ) + 1;

            vid_move_cursor( 21 * CHAR_XEXT, j * CHAR_YEXT );
            vid_put_string( OptionsStrings[2+atoi(pData)].pString ); /* Color */
            pData += strlen( pData ) + 1;

            vid_move_cursor( 29 * CHAR_XEXT, j * CHAR_YEXT );

```



```

        for ( k = 0; k < 6; k++ )
        {
            if ( atoi(pData) == 1 )
                vid_put_string("X      ");
            else
                vid_put_string("      ");
            pData += strlen(pData) + 1;
        } /* check the next option */
    } /* display the next record in the buffer */

    /* Either filled up a page or displayed all records */
    if ( i == query.rec_cnt )
    {
        if ( bLast )
        {
            vid_move_cursor( 10 * CHAR_XEXT, 24 * CHAR_YEXT );
            vid_put_string( "Last record is displayed." );
        }
        else
        {
            /* Get a new batch of records in, reset index to start */
            query.direction = DIRN_NEXT;
            bLast = DBCars_FetchRecords( MORE_RECORDS, &query );
            i = 0;
            pIndex = (DB_INDEX_NODE *) ( query.pBuffer + query.amt_memory
                                         - sizeof(DB_INDEX_NODE) );
        } /* read in more records */
    } /* last record was displayed on this page */
    else
    {
        vid_move_cursor( 10 * CHAR_XEXT, 24 * CHAR_YEXT );
        vid_put_string( "Page Down to see more records..." );

        do
        {
            /* wait for the page down key to be pressed */
            event_read( &event );
        } while ( ! (event.msg == EVENT_CHAR && event.param == EC_PAGE_DOWN) );
    } /* there will be another page of records to display */
} while ( bLast == DM_ERROR );

} /* end of Cars Report function */

int DBCars_FetchRecords( type_fetch, pQuery )
int type_fetch;
db_query *pQuery;
{
    int    erc, ret = DM_ERROR;

    erc = db_mgr( type_fetch, pQuery );

    switch( erc )
    {
        case DB_NO_ROWS_SELECTED:
        case DB_NO_RECORDS:
            /* There are no records to display in the report */
            DBCars_DisplayMsg("Report is empty, no records to display");
            ret = DM_ERROR;
            break;

        case DB_FIRST_RECORD:
            /* First check for only one record in the buffer */
            if ( pQuery->rec_cnt == 1 )
                ret = TRUE;
            else
            {
                /* only returned when the direction is DIRN_PREV/DIRN_LAST */
                /* this program should never get this return code */
                DBCars_DisplayMsg("Database returned FIRST_RECORD");
                ret = DM_ERROR;
            }
            break;

        case DB_LAST_RECORD:
            /* Returned when the direction is DIRN_NEXT/DIRN_FIRST */
            ret = TRUE;
            break;
    }
}

```



```

        case DB_OK:
            /* Only one record fit in the buffer */
            if ( pQuery->rec_cnt == 1 )
                ret = TRUE;
            break;

        default:
            /* A real error occurred during the record fetching */
            DBCars_DisplayError( err );
            ret = DM_ERROR;
            break;
    } /* end of switch on error code returned by fetch */

    return( ret );
} /* end of Cars Fetch Record module */

```

This example simply displays the actual database return code. Your application should display an informative message based on the error code. During development, displaying the error code is beneficial to the programmer.

```

void DBCars_DisplayError( err_no )
int    err_no;          /* Database Error Number */
{
    MSGBOX message;
    char  msg[90];
    char  nbr_buff[5];

    /* Build a message informing user of the Database error number returned */
    strcpy(msg, "Error number returned was: ");
    itoa( err_no, nbr_buff, 10 );
    strcat(msg, nbr_buff);

    message.btn_combo = MSG_COMBO_OK;
    message.pString   = "Database Error";
    message.pMessage  = msg;

    msg_run( &message );
} /* end of Cars display database error message function */

void DBCars_DisplayMsg( msg_string )
char  *msg_string;      /* General Message String */
{
    MSGBOX message;

    message.btn_combo = MSG_COMBO_OK;
    message.pString   = "Informative Message or Error";
    message.pMessage  = msg_string;

    msg_run( &message );
} /* end of Cars display message function */

```

It is very important to initialize file and table handles to negative values since 0 (zero) is a valid handle. Likewise the record number should also be initialized to a negative number.

```

void DBCars_SetUntitled( pDatafile )
DB_DATAFILE *pDatafile;
{
    register int i;

    /* Reset variables in the Database File structure */
    pDatafile->Filename[0] = '\0';
    pDatafile->TmpFilename[0] = '\0';
    pDatafile->hFile        = DM_ERROR;
    pDatafile->hTable       = DM_ERROR;
    pDatafile->rec_num      = DM_ERROR;

    /* Disable the Model and View menus completely,
    /* only the File options are available in this state */
    for ( i = 0; i < NUM_MODEL_MENU_ITEMS; i++ )
        ModelMenuItem[i].bEnabled = DISABLED;

    for ( i = 0; i < NUM_VIEW_MENU_ITEMS; i++ )
        ViewMenuItem[i].bEnabled = DISABLED;
}

```



```

    /* Draw the application menubar in the base window */
    CarsMENUBAR.bRedraw = MB_NO_REDRAW;
    mb_draw(&CarsMENUBAR);
} /* end of set untitled application state function */

void DBCars_SetNoRecords()
{
    register int    i;

    /* Disable the Model (except for add) and View menus completely */
    ModelMENUITEM[0].bEnabled = ENABLED;
    ModelMENUITEM[1].bEnabled = DISABLED;
    ModelMENUITEM[2].bEnabled = DISABLED;

    for ( i = 0; i < NUM VIEW MENU ITEMS; i++ )
        ViewMENUITEM[i].bEnabled = DISABLED;

    /* Draw the application menubar in the base window */
    CarsMENUBAR.bRedraw = MB_NO_REDRAW;
    mb_draw(&CarsMENUBAR);
} /* end of set no records application state function */

/* end of dbcars.c */

```


Page Printing - DEVICE.PDM

DEVICE.PDM is a template DeskMate application designed to demonstrate how the application programmer utilizes the DeskMate high level print routines. These routines are identified by beginning with "ptd_". See the DeskMate Technical Reference Print Manager for a complete list of these high level Device calls. The source for the Device Print application is included in the SAMPLES\PRINT\DEVICE directory.

*/

This application was written from the file I/O examples.

```
#include "csrprt.h"      /* Core Services Resource printer header file */
#include "csrcfg.h"      /* Core Services Resource configuration header file */
#include "device.h"      /* Application header file */
#include "devicdec.h"    /* Applications function declarations */
```

```
extern int dmerrno;
```

One of the first things this application does is initialize DeskMate's internal structures for page setup and page mode. These are defined in the .H file. Then a check is performed via an application subroutine that determines if the print menu option on the file menu should be grayed. Just as in the File I/O examples the command line is checked for a filename to be opened. This application uses the high level `fil_menu_*` calls. The main event loop is then entered.

```
int main( argc, argv )
int argc;
char *argv[];
{
    int      Done;
    int      TSReturnCode;
    int      FMReturnCode;
    int      DPRReturnCode;

    unsigned int      TempFileSize;
    unsigned int      OldFileSize;

    /* set DeskMate's current page setup information from my structures */
    ptd_set_page( &DevicePGSETUP, &DevicePGMODE );

    /* check to see if the Print menu item needs to be grayed */
    Device_Check_For_Print_Graying();

    /* Check to see if a filename was passed */
    /* to this program on the command line */
    if( argc > 1 )
    {
        /* open and load the validated file name file */
        DeviceDATAFILE.FileSize = fil_menu_open( &DeviceDATAFILE,
                                                OPEN_NO_DIALOG );
    }
    else
    {
        /* disable save menu item because there */
        /* is no file currently in memory */
        FileMenuItems[SAVE_INDEX].bEnabled = DISABLED;
    }

    /* Draw the main screen and */
    /* clear the it first */
    Device_Draw_Screen( CLEAR );

    /* initialize the do while control flag */
    Done = FALSE;

    /* Process the user inputs and actions */
    do
    {
        /* read an event from an input device */
        event_read( &Event );
```



```

switch( Event.msg )
{
    case EVENT_COMMAND :
        /* check to see if an item was selected from the menu bar */
        /* process menu item that was selected */

        switch( Event.param )
        {

```

The Page setup request runs DeskMate's page setup accessory. After the accessory is run the page setup and page mode structures must be internally initialized, and the screen is redrawn.

```

        case FILE_PAGE_SETUP_ID:
            /* the user wants to run page setup */
            fil_menu_page();

            /* get the current page setup information from */
            /* DeskMate into my page setup structures */
            ptd_get_page( &DevicePGSETUP, &DevicePGMODE );

            /* redraw the current file status information */
            /* clear from the menu bar down first */
            Device_Draw_Status_Info( CLEAR );
            break;

```

Device_Print_DataFile is an application defined subroutine which prints the data currently loaded in memory.

```

        case FILE_PRINT_ID:
            /* the user wants to print the current file */
            DPReturnCode = Device_Print_DataFile();
            if( DPReturnCode == TRUE )
            {
                /* The data printed successfully */
            }
            else
            {
                /* The data could not be printed */
            }

        case EVENT_APPL :
            switch( Event.param )
            {
                /* check for an accessory event */
                case APPL_ACCESS:
                    /* run the requested accessory */
                    dm_acc_run ( Event.x );

```

After an accessory is run it is necessary to check the printer information because if the setup accessory is run the printer may be disabled or changed, so a check is made to determine if the print option on the file menu should be grayed

```

                    /* setup could have been the accessory that was */
                    /* run so we need to check the printer setup info */
                    Device_Check_For_Print_Graying();

                    /* redraw the screen when the accessory is finished */
                    /* but do not clear the screen first */
                    Device_Draw_Screen( CLEAR );

                    break;

                /* check for a task switch event */
                case APPL_TASK_SWITCH:

```

Before execution of a Task Switch the application must save the current page setup information for this application. The reason for this is so that the information can be reset when returning from the Task Switch operation.

```

                    /* get the current page setup information from */
                    /* DeskMate into my page setup structures so that */
                    /* I can reset the information upon returning */
                    ptd_get_page( &DevicePGSETUP, &DevicePGMODE );

```



```

/* dm_yield is the call to allow task switch to occur */
TSReturnCode = dm_yield();

if ( TSReturnCode == DM_NOT_ALLOWED )
    /* task switching not being allowed by DESK.EXE */
    break;

```

The task switch was successful so redraw the screen, set the page setup information with the information that was received on the `ptd_get_page`, before the Task Switch occurred. The Task Switch'ed application may have altered the page setup information. It is the responsibility of each application to set the page setup before trying to use it.

```

if ( TSReturnCode == DM_OK )
{
    /* The task switch has occurred so */
    /* redraw the menubar and the screen */
    Device_Draw_Screen( NO_CLEAR );
    /* reset DeSkMate's current page setup */
    /* information from my structures */
    ptd_set_page( &DevicePGSETUP, &DevicePGMODE );
}

```

`Device_Draw_Screen` functions similar to all of the other example applications. The only difference here is a flag to determine whether to clear the screen before redrawing the applications main default screen.

```

void Device_Draw_Screen( bClear )
int bClear;
{
    if( bClear == TRUE )
    {
        /* Clear the base window (defaults to the entire screen) */
        vid_clear_screen();
    }

    /* Draw the application menubar in the base window */
    DeviceMENUBAR.bRedraw = MB_REDRAW;
    mb_draw( &DeviceMENUBAR );

    /* Display the application's name on the title line */
    ttl_put_app_name("Device Print");

    /* Display the application's data file name on the title line */
    /* Sending a pointer to a null string will display "Untitled" */
    ttl_put_data_name( DeviceDATAFILE.pFilename );

    Device_Draw_Status_Info( NO_CLEAR );
}

```


Device_Print_DataFile is an application subroutine which prints the data in memory. The first thing is to prompt the user for the device to be printed to. The device is then opened unless cancel is returned from the device to print to dialog box. The menu bar is erased if the user selected "print to screen". The working variables are initialized so that the page to be printed will be built with the correct line width and printed lines per page. Landscape and portrait are the only modes supported. Each page is then built in memory and sent to the printer. The page is then printed, if TRUE is sent then the last page is printed. The prompt between pages is automatically handled by DeskMate in these high level page printing functions. When the file has finished printing the menu bar is redrawn if necessary and the device is closed.

```
int Device_Print_DataFile()
{
    int PTDReturnCode;
    int Done;
    int ErasedMenuBar;
    int LineCount;
    int CharCounter;
    char *CurrentChar;
    int LineWidth;
    int PrintedLinesPage;
    int ClosePTD;
    int BufferOffset;
    int CharsRemaining;

    /* prompt the user for a device to print to */
    /* allow all devices to be printed to */
    PTDReturnCode = ptd_open( PTD_DEVICES );
    if( PTDReturnCode == PTD_CANCEL )
        return( FALSE );

    /* the device is now open */
    /* don't forget to close it */
    /* when exiting, unless CANCEL */
    ClosePTD = TRUE;

    /* check to see if "print to screen" has been */
    /* selected, if so we need to erase the menu bar */
    if( PTDReturnCode == PTD_TO_SCREEN )
    {
        mb_erase();
        ErasedMenuBar = TRUE;
    }
    else
        ErasedMenuBar = FALSE;

    if( DevicePGSETUP.mode == LANDSCAPE )
    {
        /* initialize w/landscape values */
        PrintedLinesPage = DevicePGSETUP.mLandscape.plinepp;
        LineWidth = DevicePGSETUP.mLandscape.lnwidth;
    }
    else
    {
        /* ASSUME PORTRAIT */
        /* initialize LineWidth, PrintedLinesPage */
        PrintedLinesPage = DevicePGSETUP.mPortrait.plinepp;
        LineWidth = DevicePGSETUP.mPortrait.lnwidth;
    }

    /* if the file is a DeskMate file, do not print the header info */
    /* set the offset before entering the main print loop */
    if( DeviceDATAFILE.FileType != CSR_NULL )
        BufferOffset = FILE_HEADER_LENGTH;
    else
        BufferOffset = 0;

    /* initialize CurrentChar */
    CurrentChar = DeviceDATAFILE.pStart + BufferOffset;

    /* no lines, characters or pages have been printed yet */
    LineCount = 0;
    Done = FALSE;
}
```



```

/* print a page */
while ( Done == FALSE ) /* while not done */
{
    /* initialize a new page */
    ptd_start_page( CSR_ERROR );

    /* print all lines on a page */
    /* keep printing a page until the number of lines printed */
    /* is equal to the number of printed lines per page */
    while( LineCount < PrintedLinesPage )
    {
        /* check to see if we are at the end of memory */
        if( CurrentChar + LineWidth > DeviceDATAFILE.pEnd - 1 )
        {
            /* there is less than 1 line left */
            CharsRemaining = DeviceDATAFILE.pEnd - CurrentChar;
            ptd_put_nchars( CurrentChar, CharsRemaining );
            Done = TRUE;
            break;
        }

        PTDReturnCode = ptd_put_nchars( CurrentChar, LineWidth );
        if( PTDReturnCode == CSR_ERROR )
        {
            /* the page list is full */
            break;
        }

        CurrentChar += LineWidth;
        LineCount++;
    } /* finished building a page */

    /* print the page we just built */
    /* if it is the last page send TRUE */
    if( Done == TRUE )
        PTDReturnCode = ptd_print_page( TRUE );
    else
        PTDReturnCode = ptd_print_page( FALSE );
    if( PTDReturnCode == CSR_ERROR )
    {
        Done = TRUE;
        ClosePTD = FALSE;
    }
    LineCount = 0;
}

if( ErasedMenuBar == TRUE )
{
    /* Draw the application menubar in the base window */
    DeviceMENUBAR.bRedraw = MB_REDRAW;
    mb_draw( &DeviceMENUBAR );
}
else
    if( ClosePTD == TRUE )
        ptd_close();
}

```

Device_Check_For_Print_Graying checks to see if a printer driver is loaded. Then checks to make sure the printer is text and that it will print in portrait mode. Finally a check is done to see if there is a datafile to print. If there is no datafile, print must be disabled.

```

void Device_Check_For_Print_Graying()
{
    /* get the printer configuration information */
    cfg_get_prt_data( &DevicePRINTER_CFG );
    if( DevicePRINTER_CFG.driver[0] == '\0' )
        FileMenuItems[PRINT_INDEX].bEnabled = DISABLED;
    else
    {
        /* there is a driver loaded */
        prt_get_printer( &DevicePRINTER );
        if( DevicePRINTER.bTextOnly == TRUE && DevicePGSETUP.mode != PORTRAIT )
            FileMenuItems[PRINT_INDEX].bEnabled = DISABLED;
        else
            FileMenuItems[PRINT_INDEX].bEnabled = ENABLED;
    }
}

```



```
/* gray the print menu item if there is no file to print */  
if( DeviceDATAFILE.pStart == DeviceDATAFILE.pEnd )  
    FileMenuItems[PRINT_INDEX].bEnabled = DISABLED;  
}
```


Direct Printing - DIRECT.PDM

DIRECT.PDM is a template DeskMate application designed to demonstrate how the application programmer utilizes the DeskMate low level print routines. This application utilizes the various low level or Direct print routines which DeskMate provides to an application. These routines are identified by starting with "prt_". See the DeskMate Technical Reference Print Manager for a complete list of these low level Direct calls. This application only prints in portrait mode. The source for this Device Print application is included in the SAMPLES\PRINT\DIRECT directory.

```
#include "csrcfg.h"      /* Core Services Resource configuration header file */
#include "direct.h"      /* Application header file */
#include "direcdec.h"    /* Application function declarations */

extern int dmerrno;

int main( argc, argv )
int argc;
char *argv[];
{
    int      Done;
    int      TSReturnCode;
    int      FMReturnCode;
    int      DPReturnCode;

    unsigned int    TempFileSize;
    unsigned int    OldFileSize;
```

The page setup getting and setting are the same as the high level Device printing as explained in the previous section.

```
case FILE PAGE SETUP ID:
    /* the user wants to run page setup */
    fil_menu_page();

    /* get the current page setup information from */
    /* DeskMate into my page setup structures */
    ptd_get_page( &DirectPGSETUP, &DirectPGMODE );

    /* redraw the current file status information */
    /* clear from the menu bar down first */
    Direct_Draw_Status_Info( CLEAR );
    break;
```

Direct_Print_DataFile is an application subroutine which prints the data from memory to the printer.

```
case FILE PRINT ID:
    /* the user wants to print the current file */
    DPReturnCode = Direct_Print_DataFile();
    if( DPReturnCode == TRUE )
    {
        /* The data printed successfully */
    }
    else
    {
        /* The data could not be printed */
    }

    /* redraw the current file status information */
    Direct_Draw_Status_Info( NO_CLEAR );
    break;
.
.
}
```


Direct_Print_DataFile is the application routine that prints the data in memory to the printer. The first thing it does is open the print device (no dialog box is displayed). Then a dialog box is displayed giving the user the option to cancel the print operation. This was automatic in Device printing. Then a check is made to the event queue to see if the user wants to cancel printing (any event is assumed to be a cancel request). Then all of the lines are printed on a page until the lines printed are equal to the Portrait lines per page.

```
int Direct_Print_DataFile()
{
    int PRTReturnCode;
    int Done;
    int LineCount;
    int CEReturnCode;
    int CharCounter;
    char *CurrentChar;
    register int BufferOffset;
    register int CharOffset;

    /* open the direct print device LPT1, LPT2 or LPT3 */
    PRTReturnCode = prt_open( );
    if( PRTReturnCode == CSR_ERROR )
        return( FALSE );

    /* the device is now open */

    /* inform the user that we are about to print */
    /* this also gives the user the chance to cancel printing */
    Msg.pMessage = "Printing in progress...";
    Msg.pString = "Printing";
    Msg.btn combo = MSG_COMBO_CANCEL;
    msg_draw( &Msg );

    {
        /* check to see if the user wants to stop printing */
        CEReturnCode = Direct_Check_Event();
        if( CEReturnCode != CSR_NULL )
        {
            /* clear the message box */
            vid_move_cursor( 0, 5 * CHAR_YEXT );
            vid_clear_to_bot();
            /* remove the event from the event que */
            event_purge();
            Direct_Print_Form_Feed( LineCount );
            prt_close();
            return( FALSE );
        }

        /* print all lines on a page */
        /* keep printing a page until the number of lines printed */
        /* is equal to the number of printed lines per page */
        while( LineCount < DirectPGSETUP.mPortrait.plinepp )
        {
            /* print the left margin on each line */
            for( CharCounter = 0; CharCounter <= DirectPGSETUP.mPortrait.left;
                CharCounter++ )
            {
                /* put out a space */
                PRTReturnCode = prt_put_char( ' ' );
                if( PRTReturnCode == CSR_ERROR )
                {
                    prt_close();
                    return( FALSE );
                }
            }
        }
    }
}
```



```

/* print a single line on a page */
BufferOffset += CharOffset;
CharOffset = 0;
CharCounter = 0;
for( ; CharCounter < DirectPGSETUP.mPortrait.lnwidth; CharOffset++ )
{
    CurrentChar = DirectDATAFILE.pStart + (BufferOffset+CharOffset);

    /* check to see if the user wants to stop printing */
    CEReturnCode = Direct Check Event();
    if( CEReturnCode != CSR_NULL )
    {
        /* clear the message box */
        vid move cursor( 0, 5 * CHAR_YEXT );
        vid clear to bot();
        /* Remove the event from the event que */
        event purge();
        Direct Print Form_Feed( LineCount );
        prt close();
        return( FALSE );
    }

    /* check to see if we are at the end of memory */
    if( CurrentChar > DirectDATAFILE.pEnd - 1 )
    {
        Direct Print Form_Feed( LineCount );
        prt close();
        Done = TRUE;
        return( TRUE );
    }
}

```

prt_put_char prints a character to the device. **prt_put_char** filters out certain control codes, any non-ascii characters with the exception of 10h, 11h, 12h, and 13h will not be printed. Applications may want to enter special checking for carriage returns and line feeds here, but this SIMPLE demo does NOT respect those particular control characters.

```

/* *(Current Char) -- Print a single character on the device.*/
PRTReturnCode = prt put char( *(Current Char) );
if( PRTReturnCode == CSR_ERROR )
{
    prt close();
    return( FALSE );
}
/* since CR's and LF's are not printable, they should not */
/* be included as part of the character count */
if( *(CurrentChar) == 0x0d ||
    *(CurrentChar) == 0x0a )
    ; /* --CharCounter; */
else
    /* the character was successfully printed */
    /* increment the character counter */
    ++CharCounter;

} /* finished printing a line */

/* At end of line, put a CR. */
/* This was automatic in High Level Device. */
/* put out a carriage line */
prt put tty( '\r' );
if( DirectPRINTER_CFG.bAutoLF == CR_ONLY ||
    DirectPRINTER.bTextOnly == PRT_TEXT_ONLY )
    /* put out a line feed */
    prt put tty( '\n' );
LineCount++;

} /* finished printing a page */

Direct Print Form_Feed( LineCount );
LineCount = 0;
}
}

```


Direct_Check_Event scans for any event in the event queue. This routine was done automatically in the high level Device printing.

```
int Direct_Check_Event()
{
    event_scan( &Event );
    if( Event.msg != CSR_NULL )
        return( Event.msg );
    else
        return( CSR_NULL );
}
```

Direct_Print_Form_Feed prints the appropriate number of blank lines on the page so that the paper will be at the top of the form. This function was automatic in the high level Device printing.

```
void Direct_Print_Form_Feed( LineCount )
int LineCount;
{
    while( LineCount < DirectPGSETUP.mPortrait.linepp )
    {
        prt_put_tty( '\x0d' );
        if( DirectPRINTER_CFG.bAutoLF == CR_ONLY ||
            DirectPRINTER.bTextOnly == PRT_TEXT_ONLY )
            prt_put_tty( '\x0a' );
        LineCount++;
    }
}
```


FORMS.PDM

FORMS.PDM is a DeskMate program which uses the Form Manager to manage, display, and print graphic shapes. This example uses a "working window" to display the graphics. Refer to the Special Topics section for more information about Managing Windows and Events.

```
#include "csrform.h" /* CSR form manager header file */
#include "dmfont.h" /* form font definitions */
#include "dmfontfrm.h" /* font structure definitions */
#include "forms.h" /* Application header file */
#include "formdecs.h" /* App Prototype header file */
#include "fontbox.h" /* App structure definitions */

main()
{
    register FORM *pForm; /* Pointer to allocated graphics form */
    register FORM_HDR *pFormHdr; /* Pointer to graphics form header */
    EVENT Event;
    int Done, bMode;
    int oldx,oldy,xdif,ydif;
    int mem_available;
    MAPRECT map,region;
```

The enhanced form manager with font support is a separate resource which must be requested by the application.

```
if ( eform_bind_init(1) == CSR_ERROR )
{
    /* failure to bind to the Enhanced Form Manager */
    Forms_DisplayMsg("Enhanced Form Resource could not be loaded");
    guf_bind_end();
    csr_end();
    exit(1);
}
```

Get the largest block of memory available, but leave 50k of memory free for printing purposes.

```
/* setup buffers in font engine ( get all memory available) */
mem_available =font_set_buffer(-1); /* get largest block of memory */

/* leave 50k of memory free to print */
/* if largest block is less than 4k then exit */
if ( (mem_available = mem_available - (50*1024)/16) < 256 )
{
    Forms_DisplayMsg("Not able to get enough memory");
    guf_bind_end();
    eform_bind_end();
    csr_end();
    exit(1);
}
else
font_set_buffer(mem_available);
```

The application allocates the data space for the graphics form. The data space MUST be contiguous.

```
/* Allocate the graphics form */
pForm = (FORM *) malloc( sizeof(FORM) );
if ( pForm == NULL )
{
    /* always free resources before exiting program */
    Forms_DisplayMsg("Insufficient memory to allocate graphics form.");
    eform_bind_end();
    guf_bind_end();
    csr_end();
    exit(1);
}
```



```

/* Setup the graphics form, remainder of information is */
/* supplied by the Form Manager when the form is opened */
pFormHdr = &(pForm->header);

pFormHdr->bNewList      = NEW FORM;
pFormHdr->bVideo        = TRUE;
pFormHdr->list_size     = SIZE_OF_LIST;
pFormHdr->stroke_size  = SIZE_OF_STROKE;
eform_open((FORM_HDR far *) pFormHdr );

/* Set the Mode to Draw, no element is currently selected */
bMode = DRAW;
pForm->pElement = NULL;
pForm->tag = CSR_NULL;

```

All applications which use the Page Setup function should register their default page setup information with the CSR before any printing is done.

```

/* Register the default page setup information for printing */
ptd_set_page( &FormsPGSETUP, &FormsPGMODE );

/* Draw the main screen, switch to the child window */
Forms_DrawScreen( pFormHdr, pForm->pElement );

/* initialize the do while control flag */
Done = FALSE;

```

The Event processing here is very similar to that seen in the other programming examples except in this example the event is passed on to another function which processes the event. Mouse and keyboard events are processed within the event processing loop.

```

/* Process the user inputs and actions */
do
{
    /* read an event from an input device */
    event_read(&Event);

    switch( Event.msg )
    {

```

The CHAR processing determines what element has been selected when the keyboard is used.

```

        case EVENT_CHAR :
        ...

```

The MOUSE processing determines what element has been selected when the mouse is used.

```

        case EVENT_MOUSE :
        ...

```

The OUTSIDE processing determines if an event outside the current active window has taken place.

```

        case EVENT_OUTSIDE :
        ...

```

The APPL determines if a special CSR defined event has taken place.

```

        case EVENT_APPL :
        ...

    } /* end of switch on type of event */

}
/* check to see if "EXIT" or "RUN" menu item has been selected */
while( Done != TRUE );

```


When EXIT or RUN has been selected we leave the event loop and exit the application after releasing the loaded resources.

```
/* release the loaded resources before exiting */
eform_bind_end();
guf_bind_end();
csr_end();
exit(0);

} /* end of Forms main module */
```


This function sets the Edit Menu appropriately, draws the application menu bar and displays the graphics form in the working window.

```
void Forms_DrawScreen( pFormHdr, pElement )
FORM_HDR   *pFormHdr;
ELEMENT    *pElement;
{
    register int i;
    int      type, length;
    char far *lpBuffer;

    /* Check for graphics on the clipboard, set the Edit menu */
    dm_get_clipboard_info( (int far *) &type, (int far *) &length,
                           (char far **) &lpBuffer );
    if ( type == CLIP_DRAW )
    {
        /* Enable Paste, Disabled Cut, Copy, and Clear */
        EditMENUITEM[EDIT_PASTE_INDEX].bEnabled = ENABLED;
        EditMENUITEM[EDIT_CUT_INDEX].bEnabled   = DISABLED;
        EditMENUITEM[EDIT_COPY_INDEX].bEnabled  = DISABLED;
        EditMENUITEM[EDIT_CLEAR_INDEX].bEnabled = DISABLED;
    } /* clipboard contains Draw graphics */
    else
    {
        /* Disable Cut, Copy, Paste, and Clear */
        for ( i = 0; i < EDIT_COUNT - 1; i++ )
            EditMENUITEM[i].bEnabled = DISABLED;
    } /* this application doesn't handle the clipboard type */

    /* Draw the application menubar in the base window */
    if ( win_get_active() != hBase )
        win_activate( hBase );
    FormsMENUBAR.bRedraw = MB_REDRAW;
    mb_draw(&FormsMENUBAR);
    FormsMENUBAR.bRedraw = MB_NO_REDRAW;

    /* Display the application's name on the title line */
    ttl_put_app_name("Forms Manager");

    /* Display the application's data file name on the title line */
    /* Sending a pointer to a null string will display "Untitled" */
    ttl_put_data_name("Graphics Example");

    /* Draw the graphics, clearing the screen first */
    win_activate( hChild );

    /* Draw the current form on the screen */
    Forms_UpdateScreen( pFormHdr, pElement );
} /* end of form draw screen module */

void Forms_UpdateScreen( pFormHdr, pElement )
FORM_HDR   *pFormHdr;
ELEMENT    *pElement;
{
    /* erase the cursor during the updating of the screen */
    vid_erase_cursor();
    eform_update( (FORM_HDR far *) pFormHdr, ENABLED );

    /* Highlight the currently selected item (if there is one) */
    if ( pElement != NULL )
        Forms_DrawBox( pElement );

    vid_move_cursor( 0, 0 );
    /* turn the cursor back on */
    vid_draw_cursor();
}
```


The Forms_AddShape call is made to add the appropriate type element to the graphics form. The data structure for the element is filled in and the function call made.

```
void Forms_AddShape( pFormHdr, command )
FORM_HDR *pFormHdr;
int command;
{
    ELEMENT *pElement;
    EVENT event;
    register int index, i;
    int x, y, xl, yl;
    int val;
    char ch;
    char string_buffer[80];

    vid_read_cursor( &x, &y );

    switch( command )
    {
        case SHAPES_LINE_ID:
            /* add a line element to the form */
            pElement = (ELEMENT *) &FormsFORM_LINE;
            pElement->x0 = x;
            pElement->y0 = y;
            pElement->x1 = x + 10 * CHAR_XEXT;
            pElement->y1 = y + 5 * CHAR_YEXT;
            index = SHAPES_LINE_INDEX;
            break;

        case SHAPES_RECT_ID:
            /* add a rectangle element to the form */
            pElement = (ELEMENT *) &FormsFORM_RECT;
            index = SHAPES_RECT_INDEX;
            pElement->x0 = x;
            pElement->y0 = y;
            pElement->x1 = x + 15 * CHAR_XEXT;
            pElement->y1 = y + 5 * CHAR_YEXT;
            break;

        case SHAPES_ELLIPSE_ID:
            /* add a ellipse element to the form */
            pElement = (ELEMENT *) &FormsFORM_ELLIPSE;
            index = SHAPES_ELLIPSE_INDEX;
            pElement->x0 = x;
            pElement->y0 = y;
            pElement->x1 = x + 12 * CHAR_XEXT;
            pElement->y1 = y + 6 * CHAR_YEXT;
            break;

        case SHAPES_TEXT_ID:
            /* add a text element to the form */
            pElement = (ELEMENT *) &FormsFORM_TEXT;
            index = SHAPES_TEXT_INDEX;
            FormsFORM_TEXT.pString = "This is a text element.";
            FormsFORM_TEXT.nChars = strlen(FormsFORM_TEXT.pString);
            pElement->x0 = x;
            pElement->y0 = y;
            pElement->x1 = x + FormsFORM_TEXT.nChars * CHAR_XEXT;
            pElement->y1 = y + CHAR_YEXT;
            break;

        case SHAPES_FONT_ID:
            /* add a font element to the form */
            pElement = (ELEMENT *) &FormsFORM_FONT;

            eform_update( (FORM_HDR far *) pFormHdr, ENABLED );
            ms_draw_pointer();
            /* fill in the element structure for a font */
            FormsFORM_FONT.info.fpFaceInfo =
                fpFontFaceInfo+font_base+font_selected;
            FormsFORM_FONT.info.fpFaceInfo->bStyleAttrs = bStyle;
            FormsFORM_FONT.info.string_xorg = x;
            FormsFORM_FONT.info.string_yorg = y;
            FormsFORM_FONT.info.fgnd_color = COLOR2;
            FormsFORM_FONT.u_header.element.type = FORM_OTHER;
            /* fonts are assigned OTHER */
            FormsFORM_FONT.u_header.element.mod = FORM_U_FONT_TYPE;
    }
}
```



```

vid_busy_disable();
vid_move_cursor(0,0);
vid_put_string("String:");
vid_clear_to_eol();
val=0;
event_read(&event);
/* get a string */
while ( event.msg == EVENT_CHAR && event.param != 0x0d &&
event.param != 0x16 )
{
    ch = (char) event.param;
    if ( ch == 0x08 )
    {
        vid_read_cursor(&x1,&y1);
        x1 = x1 - CHAR_XEXT;
        vid_move_cursor(x1,y1);
        vid_put_char(' ');
        vid_move_cursor(x1,y1);
        val--;
    }
    else
    {
        vid_put_char(ch);
        string_buffer[val]=ch;
        val++;
    }
    event_read(&event);
}
vid_busy_enable();
if ( event.param != 0x16 )
{
    /* fill in the FORM_FONT structure (element structure) */
    FormsFORM_FONT.info.fpString = (char far *) string_buffer;
    FormsFORM_FONT.info.fpEscapement = 0;
    FormsFORM_FONT.info.nChars = val; /* string size */
    /* get a bounding box for the font string. Fills in the */
    /* maprect. This is VERY IMPORTANT */
    evid_get_bounding_box((FORM_FONT far *) &FormsFORM_FONT);
    break;
}
return;

default:
break;

} /* end of switch on shape being added */

for ( i = 0; i < SHAPES_COUNT; i++ )
/* Check the shape selected, uncheck all the others */
ShapesMENUITEM[i].bChecked = (i == index) ? MB_CHECKED :
MB_UNCHECKED;
/* add the element to the form */
if ( eform_add_element((FORM_HDR far *) pFormHdr, (ELEMENT far *)
pElement) == CSR_ERROR )
Forms_DisplayMsg("Error adding graphics element to list.");
else
/* Enable printing - we know we have graphics in the form */
FileMENUITEM[FILE_PRINT_INDEX].bEnabled = ENABLED;

/* Update menu bar information without actually redrawing it */
mb_draw(&FormsMENUBAR);

/* Move the cursor to the origin of the new shape */
vid_move_cursor( x, y );

} /* end of add shape function */

```


The `Forms_Select` demonstrates a typical method for determining if the user "clicked" on a graphics shape on the screen or positioned the text cursor and pressed the space bar or ENTER to "select" the graphics shape. The graphics select "handle box" is drawn around the shape when it is selected to show selection.

```
int Forms_Select( pForm, pEvent )
FORM      *pForm;
EVENT     *pEvent;
{
    FORM_HDR *pFormHdr;

    /* determine if there is an element at the current */
    /* cursor location or where the user clicked. */
    if ( pEvent->msg == EVENT_CHAR )
        vid_read_cursor( &pEvent->x, &pEvent->y );

    pFormHdr = &(pForm->header);
    pForm->tag = eform_find_element((FORM_HDR far *)pFormHdr, pEvent->x,
        pEvent->y, 1 );

    if ( pForm->tag > CSR_NULL )
    {
        /* First unselect a previously selected element */
        if ( pForm->pElement != NULL )
            Forms_ClearBox( pFormHdr, pForm->pElement );

        /* Now point to the element's information in the form */
        pForm->pElement = (ELEMENT *) eform_get_pointer( (FORM_HDR far *)
            pFormHdr, pForm->tag );

        /* Draw the handle box surrounding the element bounding box */
        Forms_DrawBox( pForm->pElement );

        /* Set the Edit menu for clipboard manipulation */
        EditMENUITEM[EDIT_CUT_INDEX].bEnabled = ENABLED;
        EditMENUITEM[EDIT_COPY_INDEX].bEnabled = ENABLED;
        EditMENUITEM[EDIT_CLEAR_INDEX].bEnabled = ENABLED;
        EditMENUITEM[EDIT_PASTE_INDEX].bEnabled = DISABLED;
        mb_draw( &FormsMENUBAR );
        return(1);
    } /* found a graphics element at the cursor/mouse location */
    else
    {
        /* First unselect a previously selected element */
        if ( pForm->pElement != NULL )
            Forms_ClearBox( pFormHdr, pForm->pElement );

        /* Clear pointer to currently selected element */
        pForm->pElement = NULL;

        /* Now position cursor at the stored mouse coordinates */
        vid_move_cursor( pEvent->x, pEvent->y );
        return(0);
    } /* there isn't anything at the cursor location */
} /* end of forms select graphics module */
```

This routine draws the "handle box" to show selection. Refer to the Video programming example for more information about the video functions used to draw the actual box.

```
void Forms_DrawBox( pElement )
ELEMENT     *pElement;
{
    /* First draw the rectangle surrounding the element's bounding box */
    vid_set_line_attr( LINE_SOLID, LINE_WIDTH2, COLOR_XOR );
    vid_draw_rect( pElement->x0, pElement->y0,
        pElement->x1, pElement->y1, VID_NO_FILL );

    vid_draw_rect( vid_prevn_nwcx(pElement->x1, 3), /* bottom/right corner */
        vid_prevn_nwcy(pElement->y1, 3),
        vid_nextn_nwcx(pElement->x1, 3),
        vid_nextn_nwcy(pElement->y1, 3), VID_NO_FILL );

    Cur_x = pElement->x1;
    Cur_y = pElement->y1;
}
```



```
/* Now move the cursor to the origin of the select box */  
vid_move_cursor( pElement->x0, pElement->y0 );  
} /* end of forms draw select box function */
```


The Forms_Print functions demonstrates how easy it is to do printing when using the graphics Form Manager. The Device Print Manager is used to select the printing device, initialized printing, and then print the graphics form.

```
void Forms_Print( pFormHdr, pElement )
FORM_HDR      *pFormHdr;
ELEMENT       *pElement;
{
    int          device, ret;
    PRINTER_CFG  PrtConfig;

    device = ptd_open(PTD_DEVICES);
    if ( device == PTD_TO_SCREEN )
        mb_erase();
    else if ( device == PTD_CANCEL )
    {
        Forms_DrawScreen( pFormHdr, pElement );
        return;
    }
    cfg_get_prt_data( &PrtConfig );
    if ( FormsPGSETUP.mode == LANDSCAPE )
        PrtConfig.cpi = PRT_10_CPI;

    /* Start a new page, clear the attribute flag */
    ptd_start_page( CSR_ERROR );

    /* Pointer to form, starting column, cpi at which to print form */
    ret = ptd_draw_list( pFormHdr, 0, PrtConfig.cpi );

    if ( ret == CSR_ERROR )
        Forms_DisplayMsg("Draw List call resulted in an error.");

    /* Print the form, this is the last page printed */
    ret = ptd_print_page( TRUE );

    if ( device != PTD_TO_SCREEN && ret == CSR_ERROR )
        Forms_DisplayMsg("Print Page call resulted in an error.");

    ptd_close();

    /* Redraw the application screen */
    if ( device == PTD_TO_SCREEN )
        Forms_DrawScreen( pFormHdr, pElement );
    else
        Forms_UpdateScreen( pFormHdr, pElement );
} /* end of forms print graphics module */
```


The font_selection function locates the fonts files in the system and enumerates all of the font faces.

```
font_selection()
{
    int i;
    int more_fonts ; /* return boolean code */
    int nFaces ; /* number of faces found */
    char far *tmp_fp;

    face.bType = FF_RESIDENT + FF_RASTERIZE ;
    face.request = DMF_READ ;
    nFaces = font_face_support((FACE far *) &face);

    if ((fpFontFaceInfo = (FONT_FACE far *) malloc(sizeof(FONT_FACE) *
        (nFaces - 1))) == (char *)0)
    {
        font_end() ;
        font_bind_end() ;
        csr_end() ;
        return(DMF_ERROR) ;
    }

    more_fonts = DMF_OK ;
    face.request = 0 ;
    for (i=0; (i < nFaces) && (more_fonts >= DMF_OK)); i++ )
    {
        more_fonts = font_get_face(&face, fpFontFaceInfo+i) ;
        tmp_fp = (char far *) ((fpFontFaceInfo+i)->face_name);
        FNTLB1_text_pitems[i] = (char *) (FP_OFF(tmp_fp));
    }

    font_selected = 0 ;
    FNTRBs_text[0].selected = 1 ; /*---outline, rasterize list */

    /*----present the user with the option of selecting a font face */
    if (font_select((FONT_FACE far *)fpFontFaceInfo) == (int)DMF_ERROR)
    {
        font_selected = 0 ;
        fpFontFaceInfo->PtSize = 12 ;
        fpFontFaceInfo->rotation = 0 ;
        bStyle = DFS_NORMAL ;
    }

    return(DMF_OK);
} /* end font_selection() */

int font_select(fpFontFaces)
struct font_face_defn *fpFontFaces ;
{
    int dlg_code ;
    FONT_FACE *fpFontFace ;
    FONT_FACE *f_face ;
    char ptbuf[4] ;
    char pitbuf[5] ;
    char rotbuf[4] ;

    if (FNTRBs_text[0].selected == 0)
    {
        fpFontFace = fpFontFaces ;
        FNTLBs_text[0].pitems = FNTLB1_text_pitems ;
        FNTLBs_text[0].nItems = (unsigned char)face.nResident ;
        font_base = 0 ;
    }
    else
    {
        fpFontFace = fpFontFaces + face.nResident ;
        FNTLBs_text[0].pitems = FNTLB1_text_pitems + face.nResident ;
        FNTLBs_text[0].nItems = (unsigned char)face.nRasterize ;
        font_base = face.nResident ;
    }
}
```



```

/*----set the cursor focus on first component */
FNTDlg_text.focus_index = 1;
FNTLBS_text[0].selected = (char)font_selected + 1;
f_face=(struct font_face_defn *)fpFontFace+(FNTLBS_text[0].selected-1);
FNTEFs_text[0].pBuffer = itoa(f_face->PtSize, ptbuf, 10);
FNTEFs_text[1].pBuffer = itoa(f_face->pitch, pitbuf, 10);
FNTEFs_text[2].pBuffer = itoa(f_face->rotation, rothbuf, 10);

FNTCBS_text[0].header.bEnabled =(char)((f_face->bStyle & DFS_BOLD) ?
ENABLED:DISABLED);
FNTCBS_text[1].header.bEnabled =(char)((f_face->bStyle & DFS_GRAYED) ?
ENABLED:DISABLED);
FNTCBS_text[2].header.bEnabled =(char)((f_face->bStyle & DFS_ITALIC) ?
ENABLED:DISABLED);
FNTCBS_text[3].header.bEnabled =(char)((f_face->bStyle & DFS_HOLLOW) ?
ENABLED:DISABLED);
FNTCBS_text[4].header.bEnabled =(char)((f_face->bStyle &
DFS_UNDERLINE) ? ENABLED:DISABLED);
FNTCBS_text[0].bState =(char)(bStyle & DFS_BOLD ?
CB_CHECKED:CB_UNCHECKED);
FNTCBS_text[1].bState =(char)(bStyle & DFS_GRAYED ?
CB_CHECKED:CB_UNCHECKED);
FNTCBS_text[2].bState =(char)(bStyle & DFS_ITALIC ?
CB_CHECKED:CB_UNCHECKED);
FNTCBS_text[3].bState =(char)(bStyle & DFS_HOLLOW ?
CB_CHECKED:CB_UNCHECKED);
FNTCBS_text[4].bState =(char)(bStyle & DFS_UNDERLINE ?
CB_CHECKED:CB_UNCHECKED);

if (f_face->bType == FF_RESIDENT)
{
    FNTEFs_text[0].header.bEnabled = DISABLED;
    FNTEFs_text[1].header.bEnabled = DISABLED;
    FNTEFs_text[2].header.bEnabled = DISABLED;
}
else
{
    FNTEFs_text[0].header.bEnabled = ENABLED;
    FNTEFs_text[1].header.bEnabled = ENABLED;
    FNTEFs_text[2].header.bEnabled = ENABLED;
}

/*----set the push buttons to an up position */
FNTDlg_text.return_value = CMP_NO_ACTION;
FNTPBs_text[0].bState = PB_UP;
FNTPBs_text[1].bState = PB_UP;
FNTEFs_text[0].cursor_offset = EF_SELECT_ALL;
FNTEFs_text[1].cursor_offset = EF_SELECT_ALL;
FNTEFs_text[2].cursor_offset = EF_SELECT_ALL;

dlg_draw(&FNTDlg_text);

/*----run the dialog box until user has finished */
ms draw_pointer ();
while ((FNTPBs_text[0].bState != PB_DOWN) && (FNTPBs_text[1].bState
!= PB_DOWN))
{
    dlg_code = dlg_run(&FNTDlg_text);
    if (dlg_code == (int)FNTRBI_text_tag)
    {
        if (FNTRBs_text[0].selected == 0)
        {
            fpFontFace = fpFontFaces;
            FNTLBS_text[0].pItems = FNTLB1_text_pitems;
            FNTLBS_text[0].nItems =(unsigned char)face.nResident;
            font_base = 0;
            FNTRedraw_text[1]=DLG_REDRAW;
        }
        else
        {
            fpFontFace = fpFontFaces + face.nResident;
            FNTLBS_text[0].pItems = FNTLB1_text_pitems + face.nResident;
            FNTLBS_text[0].nItems =(unsigned char)face.nRasterize;
            font_base = face.nResident;
            FNTRedraw_text[1]=DLG_REDRAW;
        }
    }
}

if (((dlg_code == (int)FNTLB1_text_tag) &&
(FNTDlg_text.return_value == (int)CMP_SELECT_CHANGE)) ||
(dlg_code == (int)FNTRBI_text_tag))

```



```

{
    f_face=(struct font_face_defn *)fpFontFace +
        (FNTLBS_text[0].selected-1) ;

    FNTCBS_text[0].header.bEnabled =(char)((f_face->bStyle &
        DFS_BOLD)? ENABLED:DISABLED) ;
    FNTCBS_text[1].header.bEnabled =(char)((f_face->bStyle &
        DFS_GRAYED) ? ENABLED:DISABLED) ;
    FNTCBS_text[2].header.bEnabled =(char)((f_face->bStyle &
        DFS_ITALIC) ? ENABLED:DISABLED) ;
    FNTCBS_text[3].header.bEnabled =(char)((f_face->bStyle &
        DFS_HOLLOW) ? ENABLED:DISABLED) ;
    FNTCBS_text[4].header.bEnabled =(char)((f_face->bStyle &
        DFS_UNDERLINE) ? ENABLED:DISABLED) ;

    if (f_face->bType == FF_RESIDENT)
    {
        FNTEFs_text[0].header.bEnabled = DISABLED ;
        FNTEFs_text[1].header.bEnabled = DISABLED ;
        FNTEFs_text[2].header.bEnabled = DISABLED ;
    }
    else
    {
        FNTEFs_text[0].header.bEnabled = ENABLED ;
        FNTEFs_text[1].header.bEnabled = ENABLED ;
        FNTEFs_text[2].header.bEnabled = ENABLED ;
    }

    FNTEFs_text[0].pBuffer = itoa((int)f_face->PtSize, ptbuf, 10);
    FNTEFs_text[1].pBuffer = itoa(f_face->pitch, pitbuf,10) ;
    FNTEFs_text[2].pBuffer = itoa((int)f_face->rotation,
        rotbuf, 10) ;
    FNTEFs_text[0].cursor_offset = EF_SELECT_ALL ;
    FNTEFs_text[1].cursor_offset = EF_SELECT_ALL ;
    FNTEFs_text[2].cursor_offset = EF_SELECT_ALL ;

    FNTRedraw_text[3]=DLG_REDRAW ;
    FNTRedraw_text[4]=DLG_REDRAW ;
    FNTRedraw_text[5]=DLG_REDRAW ;
    FNTRedraw_text[6]=DLG_REDRAW ;
    FNTRedraw_text[7]=DLG_REDRAW ;
    FNTRedraw_text[8]=DLG_REDRAW ;
    FNTRedraw_text[9]=DLG_REDRAW ;
    FNTRedraw_text[10]=DLG_REDRAW ;
}

}

ms_erase_pointer ();
event_purge() ;

vid_move_cursor(0,MB_BOTTOM_YORG-CHAR_YEXT) ;
vid_clear_to_bot();
if (FNTPBS_text[0].bState == PB_DOWN)
{
    font_selected = FNTLBS_text[0].selected-1 ;
    f_face= (struct font_face_defn *)fpFontFace+font_selected ;
    bStyle= DFS_NORMAL ;
    bStyle = FNTCBS_text[0].bState ? (bStyle|DFS_BOLD):bStyle ;
    bStyle = FNTCBS_text[1].bState ? (bStyle|DFS_GRAYED):bStyle ;
    bStyle = FNTCBS_text[2].bState ? (bStyle|DFS_ITALIC):bStyle ;
    bStyle = FNTCBS_text[3].bState ? (bStyle|DFS_HOLLOW):bStyle ;
    bStyle = FNTCBS_text[4].bState ? (bStyle|DFS_UNDERLINE):bStyle ;
    f_face->PtSize = atoi(FNTEFs_text[0].pBuffer) ;
    f_face->pitch = atoi(FNTEFs_text[1].pBuffer) ;
    f_face->rotation = atoi(FNTEFs_text[2].pBuffer) ;
    return(DMF_OK) ;
}
else
    return(DMF_ERROR) ;
}

/* end of forms.c */

```


Special Topics

This section discusses the special programming required to run components directly in the work area instead of in a dialog box, how to create and manage events from multiple windows in the work area and how to interface with the DeskMate clipboard if your application has an Edit Menu.

Running Components in the Work Area - COMPS.PDM

COMPS.PDM shows how to handle the running components in the work area of a DeskMate application. The Comps application is included in the SAMPLES\COMPS directory.

```
#include "comps.h"      /* Application header file */
#include "compsdec.h"   /* Application function declarations */
```

The application starts with the standard calls to initialize and bind to resources (Guf and the CSR). Then a call is made to draw the screen. In this case the screen is redrawn with the appropriate component drawn along with the menubar. The first time the call is made, there is no component, the handle is DM_ERROR, so no component is drawn.

```
int main()
{
    EVENT Event;
    int  TSReturnCode;
    int  Done;
    int  component = DM_ERROR;
    int  bActive;
    int  j;
    int  handle = DM_ERROR;

    /* initialize the Component_Run control flag */
    bActive = FALSE;

    /* Process the user inputs and actions */
    do
    {
        if ( bActive )
            Component_Run( component );

        /* read an event from an input device */
        event_read( &Event );

        switch( Event.msg )
        {
            case EVENT_COMMAND:
                case CMP_EDITFIELD_ID:
                    /* the user wants to see editfield operation */

                    /* set component for call */
                    component = EDITFIELD_COMPONENT;

                    /* open and draw the component */
                    handle = Component_Init(component,handle);
                    bActive = TRUE;
                    break;

                case CMP_LISTBOX_ID:
                    /* the user wants to see lisbox operation */

                    /* set component for call */
                    component = LISTBOX_COMPONENT;

                    /* open and draw the component */
                    handle = Component_Init(component,handle);
                    Component_Disable_Edit_Menuitems();
                    bActive = TRUE;
                    break;
        }
    } while ( !Done );
}
```



```

        case CMP PUSHBUTTON ID:
            /* the user wants to see radiobutton operation */

            /* set component for call */
            component = PUSHBUTTON_COMPONENT;

            /* open and draw the component */
            handle = Component_Init(component,handle);
            Component_Disable_Edit_Menuitems();
            bActive = TRUE;
            break;

        case CMP ICONBUTTON ID:
            /* the user wants to see iconbutton operation */

            /* set component for call */
            component = ICONBUTTON_COMPONENT;

            /* open and draw the component */
            handle = Component_Init(component,handle);
            Component_Disable_Edit_Menuitems();
            bActive = TRUE;
            break;

    } /* end of switch on type of application event */
    break;

} /* end of Component main module */

```

When a user selects one of the four component types from the F4 Menu, a call is made to `Component_Init` with the type of component passed as the parameter. `Component_Init` closes any component that may have been previously opened and sets the initial state of the component (the edit field cursor offset, the selected item in the list box, the state of icon and push buttons, etc.).

Any previous component which may have been on the screen is erased, the selected component is disabled from the F4 menu, and finally the selected component is drawn with a call to `cmp_draw` and the handle for the new component is returned.

If any component besides the edit field is selected, the F3 Edit menu options are all disabled. Cut, copy, paste, and clear only apply to the edit field component.

```

int Component_Init( component, handle )
int component;
int handle;
{
    int j;
    int hNewComponent;

    if ( handle != DM_ERROR )
        cmp_close( hComponent[component], pComponent[component] );

    switch (component)
    {
        case EDITFIELD_COMPONENT:
            *(ComponentEDITFIELD.pBuffer)='\0';
            ComponentEDITFIELD.cursor_offset = 0;
            break;

        case LISTBOX_COMPONENT:
            ComponentLISTBOX.selected = 1;
            break;

        case PUSHBUTTON_COMPONENT:
            ComponentPUSHBUTTON.bState = PB_UP;
            break;

        case ICONBUTTON_COMPONENT:
            ComponentICONBUTTON.bState = PB_UP;
            break;

    } /* end of switch statement */
}

```



```

/* remove previous component (if any) from screen */
vid_move_cursor( 0 * CHAR_XEXT, 3 * CHAR_YEXT );
vid_clear_to_bot();

/* enable all F4 Menu items */
for (j=0; j<4; j++)
    ComponentMENUITEM[j].bEnabled = ENABLED;

/* disable the selected F4 Menu item */
ComponentMENUITEM[component].bEnabled = DISABLED;

/* open the selected component */
hNewComponent = cmp_open( pComponent[component] );

/* draw the selected component */
cmp_draw( hNewComponent, pComponent[component] );

return( hNewComponent );
} /* end of init component */

```

After the component is initialized, control returns to the beginning of the main loop, where the component is actually run with a call to `Component_Run`. If the component is an edit field, a check is made to see if any text data is residing in the clipboard with a call to `dm_get_clipboard_info`. If text data is in the clipboard, then the "Paste" option of the F3 menu is enabled, while the other F3 options are all disabled.

While the component is running the application displays messages that show that the component is running while in `cmp_run` and the return code definition produced by the `cmp_run`.

In the case of a `CMP_SELECT_CHANGE` on an edit field component, the "Cut", "Copy", and "Clear" options of the F3 Edit menu are enabled.

```

void Component_Run(component)
int component;
{
    int ReturnCode;
    int j;
    int Done;
    char *pString;
    int Type, Length;
    char far *lpBuffer;

    /* Check if editfield is component selected. If it is and there is */
    /* something in the clipboard, then enable paste, grey cut,copy,clear... */
    /* else enable those and grey paste..... */

    if (component == EDITFIELD_COMPONENT)
    {
        /* check for text in the clipboard */
        dm_get_clipboard_info( (int far *) &Type, (int far *) &Length,
                               (char far *) &lpBuffer );

        if (Type == CLIP_TEXT)
        {
            Component Disable Edit Menuitems();
            EditMENUITEM[EDIT_PASTE_INDEX].bEnabled = ENABLED;
        }
    }
    Done = FALSE;
    do
    {
        vid_move_cursor( 26 * CHAR_XEXT, 4 * CHAR_YEXT );
        vid_put_string( "Component Running" );

        ReturnCode = cmp_run( hComponent[component], pComponent[component] );

        vid_move_cursor( 26 * CHAR_XEXT, 4 * CHAR_YEXT );
        vid_clear_to_eol();

        vid_move_cursor( 26 * CHAR_XEXT, 5 * CHAR_YEXT );
        vid_put_string( "Return code = " );
    }
}

```



```

pString = ComponentReturnStrings[ReturnCode];
vid_move_cursor( 40 * CHAR_XEXT, 5 * CHAR_YEXT );
vid_put_string(pString);

/* On pushbutton, delay, then raise the button */
if(component == PUSHBUTTON_COMPONENT)
{
    waitloop(0x20);
    ComponentPUSHBUTTON.bState = PB_UP;
    cmp_draw( hComponent[component], pComponent[component] );
}

/* On iconbutton, delay, then raise the button */
if(component == ICONBUTTON_COMPONENT)
{
    waitloop(0x20);
    ComponentICONBUTTON.bState = PB_UP;
    cmp_draw( hComponent[component], pComponent[component] );
}

switch (ReturnCode)
{
    case CMP_NO_ACTION:
        Done = TRUE;
        break;

    case CMP_CANCEL:
        break;

    case CMP_SELECT_CHANGE:
        if (Component == EDITFIELD_COMPONENT)
        {
            if (ComponentEDITFIELD.select_length > 0 )
            {
                /* only if selected text in the editfield */
                EditMENUITEM[EDIT_CUT_INDEX].bEnabled = ENABLED;
                EditMENUITEM[EDIT_COPY_INDEX].bEnabled = ENABLED;
                EditMENUITEM[EDIT_PASTE_INDEX].bEnabled = DISABLED;
                EditMENUITEM[EDIT_CLEAR_INDEX].bEnabled = ENABLED;
            }
            else
                Component_Disable_Edit_Menuitems();
        }
        break;

    case CMP_GO:
        break;

    case CMP_ACTION:
        Done = TRUE;
        break;

    case CMP_ACTION_IN_EVENT:
        break;

    default:
        break;
}
}
while (!Done);
}

void Component_Draw_Screen( component, handle )
int component;
int handle;
{
    /* draw the selected component */
    if( handle != DM_ERROR )
        cmp_draw( handle, pComponent[component] );
}

```


To reposition the cursor where the user clicked, write the event back after activating window B and have window B read it and position the cursor.

Events from the menu bar will appear as outside events to the child windows since the menu bar belongs to the base window. Remember to activate the base window BEFORE drawing the application menu bar.

```
/* Draw the application menubar in the base window */
if ( win get_active() != hBase )
    win activate( hBase );
FormsMENUBAR.bRedraw = MB_REDRAW;
mb draw(&FormsMENUBAR);
FormsMENUBAR.bRedraw = MB_NO_REDRAW;

/* Display the application's name on the title line */
ttl_put_app_name("Forms Manager");

/* Display the application's data file name on the title line */
/* Sending a pointer to a null string will display "Untitled" */
ttl_put_data_name("Graphics Example");

/* Draw the graphics, clearing the screen first */
win_activate( hChild );

/* Draw the current form on the screen */
Forms_UpdateScreen( pFormHdr, pElement );
```


Interfacing with the Clipboard

From an Editfield Component

Use the **edt_*** component utilities discussed at the end of the Component Manager section of the DeskMate Technical Reference when interfacing with the clipboard through editfields which are running directly in the editfield. In the DeskMate 3.3 system, the clipboard functions are handled automatically by the component itself if the Edit Menu accelerators are not defined and active.

```
case CUT TEXT ID:
    edt_cut( &ComponentEDITFIELD );
    cmp_draw( handle, &ComponentEDITFIELD );
    bActive = TRUE;
    break;

case COPY TEXT ID:
    edt_copy( &ComponentEDITFIELD );
    bActive = TRUE;
    break;

case PASTE TEXT ID:
    edt_paste( &ComponentEDITFIELD );
    cmp_draw( handle, &ComponentEDITFIELD );
    bActive = TRUE;
    break;

case CLEAR TEXT ID:
    edt_clear( &ComponentEDITFIELD );
    bActive = TRUE;
    break;
```

When Using the Form Manager

Use the **form_*** calls discussed in the Form Manager section of the DeskMate Technical Reference when copying to and retrieving form graphics from the clipboard. Graphics on the clipboard are complete graphics forms with header information. The entire form is one object which must be broken apart using the **form_break_object** call in order to access the individual elements in the form.

```
case EDIT CUT ID:
case EDIT CLEAR ID:
case EDIT COPY ID:
case EDIT PASTE ID:
case EDIT CLR SCRIN ID:
    Forms_EditOption( pForm, &Event );
    break;

void Forms_EditOption( pForm, pEvent )
FORM    *pForm;
EVENT   *pEvent;
{
    switch( pEvent->param )
    {
        case EDIT CUT ID:
        case EDIT CLEAR ID:
            /* Copies to clipboard & deletes it from list */
            Forms_CutElement( &(pForm->header), pForm->pElement,
                             pForm->tag, pEvent->param );
            break;

        case EDIT COPY ID:
            /* Copies element to the clipboard */
            Forms_CopyElement( &(pForm->header), pForm->tag );

            /* We don't want to leave the element selected */
            Forms_ClearBox( &(pForm->header), pForm->pElement );
            break;
    }
}
```



```

        case EDIT_PASTE_ID:
            /* Adds the element on the clipboard to the list */
            Forms_PasteElement( &(pForm->header) );
            break;

        case EDIT_CLR_SCRN_ID:
            /* First clear the graphics form, then the screen area */
            form_clear( &(pForm->header) );
            vid_move_cursor( 0, 0 );
            vid_clear_to_bot();

            /* Disable printing since we don't have any graphics */
            FileMENUITEM[FILE_PRINT_INDEX].bEnabled = DISABLED;
            mb_draw( &FormsMENUBAR );
            break;

        default:
            break;

    } /* end of switch on Edit option */

    /* Will no longer have a selected element on the screen */
    pForm->pElement = NULL;
    pForm->tag = 0;

} /* end of form edit option module */

void Forms_ClearBox( pFormHdr, pElement )
FORM_HDR    *pFormHdr;
ELEMENT     *pElement;
{
    MAPRECT    region;

    /* Redisplay the region surrounded by the select box */
    region.xorg = vid_prevn_nwcx(pElement->x0, 3);
    region.yorg = vid_prevn_nwcy(pElement->y0, 3);
    region.xext = vid_nextn_nwcx(pElement->x1, 3);
    region.yext = vid_nextn_nwcy(pElement->y1, 3);

    vid_clear_block( region.xorg, region.yorg,
                    region.xext - region.xorg + 1,
                    region.yext - region.yorg + 1 );

    form_display_region( pFormHdr, &region );
    vid_move_cursor( pElement->x0, pElement->y0 );
} /* end of forms clear select box function */

Forms_CutElement( pFormHdr, pElement, tag, command )
FORM_HDR    *pFormHdr;
ELEMENT     *pElement;
int         tag;
int         command;
{
    MAPRECT    region;
    int         ret, type, length;
    char far   *lpBuffer;

    /* Save region to remove the element from the screen */
    region.xorg = pElement->x0;
    region.yorg = pElement->y0;
    region.xext = pElement->x1;
    region.yext = pElement->y1;

    if ( command == EDIT_CUT_ID )
        /* Remove element from the graphics list & copy it to the clipboard */
        ret = form_cut_element( pFormHdr, tag );
    else
        /* Remove element from the graphics list only */
        ret = form_delete_element( pFormHdr, tag );

    if ( ret == CSR_ERROR )
        Forms_DisplayMsg("Cut or Clear operation was unsuccessful.");

    /* Clear the screen where the element was displayed */
    vid_clear_block( region.xorg, region.yorg,
                    region.xext - region.xorg + 1,
                    region.yext - region.yorg + 1 );
}

```



```

/* Redisplay the region where the element was displayed */
form display region( pFormHdr, &region );
vid_move_cursor( region.xorg, region.yorg );

/* Check for graphics in the form, if empty then disable printing */
region.xorg = 0;
region.yorg = 0;
region.xext = ChildWnd.xext;
region.yext = ChildWnd.yext;
if ( form region empty( pFormHdr, &region ) == CSR_NULL )
    FileMENUITEM[FILE_PRINT_INDEX].bEnabled = DISABLED;

/* Disable Cut, Copy, and Clear */
EditMENUITEM[EDIT_CUT_INDEX].bEnabled = DISABLED;
EditMENUITEM[EDIT_COPY_INDEX].bEnabled = DISABLED;
EditMENUITEM[EDIT_CLEAR_INDEX].bEnabled = DISABLED;

if ( command == EDIT_CUT_ID )
    /* We know there is something on the clipboard, enable paste */
    EditMENUITEM[EDIT_PASTE_INDEX].bEnabled = ENABLED;
else
{
    /* Check for graphics on the clipboard, set the Edit menu */
    dm_get_clipboard_info( (int far *) &type, (int far *) &length,
                           (char far **) &lpBuffer );
    EditMENUITEM[EDIT_PASTE_INDEX].bEnabled = ( type == CLIP_DRAW ) ?
                                                ENABLED : DISABLED;
} /* check clipboard before enabling paste */

/* Update the menu information */
mb_draw(&FormsMENUBAR);
} /* end of forms cut element function */

Forms CopyElement( pFormHdr, tag )
FORM_HDR    *pFormHdr;
int         tag;
{
    /* Copies element to the clipboard, leaves graphics list intact */
    if ( form copy element( pFormHdr, tag ) == CSR_ERROR )
        Forms_DisplayMsg("Copy operation was unsuccessful.");

    /* Enable Paste, Disabled Cut, Copy, and Clear */
    EditMENUITEM[EDIT_PASTE_INDEX].bEnabled = ENABLED;
    EditMENUITEM[EDIT_CUT_INDEX].bEnabled = DISABLED;
    EditMENUITEM[EDIT_COPY_INDEX].bEnabled = DISABLED;
    EditMENUITEM[EDIT_CLEAR_INDEX].bEnabled = DISABLED;

    mb_draw(&FormsMENUBAR);
} /* end of forms copy element function */

Forms PasteElement( pFormHdr )
FORM_HDR    *pFormHdr;
{
    int         tag, x, y;
    ELEMENT     *pElement;
    FORM_DST    Destination;

    /* past in the element from the clipboard */
    tag = form_paste( pFormHdr );

    if ( tag < 0 )
    {
        Forms_DisplayMsg("Paste operation was unsuccessful.");
        return;
    }

    /* Move the new element to the cursor location */
    /* since it was pasted at the upper-left corner */
    vid_read_cursor( &x, &y );

```



```

pElement = (ELEMENT *) form_get_pointer( pFormHdr, tag );
Destination.x0 = x;
Destination.y0 = y;
Destination.x1 = (pElement->x1 - pElement->x0) + x;
Destination.y1 = (pElement->y1 - pElement->y0) + y;
Destination.x0f = 0;
Destination.x1f = 0;
Destination.y0f = 0;
Destination.y1f = 0;

form_move_element( pFormHdr, tag, &Destination );

/* Enable printing since we now know we have some graphics */
FileMENUITEM[FILE PRINT INDEX].bEnabled = ENABLED;
mb_draw(&FormsMENUBAR);-
} /* end of forms paste element function */

```


Direct Interfacing with the Clipboard

To read the clipboard:

Source is the pointer to the clipboard area returned by **dm_get_clipboard_info**.

Dest is the pointer to the buffer area in the application data segment where the information will be copied to.

NumBytes is the size of the clipboard returned by **dm_get_clipboard_info**.

```
copy_from_clipboard ( Source, Dest, NumBytes )
unsigned char far *Source;
unsigned char *Dest;
int NumBytes;
{
    register unsigned char *d;
    register int n;

    d = Dest;
    n = NumBytes;

    for ( ; n != 0; n-- )
        *d++ = *Source++;
}
```

To write to the clipboard:

- 1) Call **dm_set_clipboard_info**, to set the type and length of your data.
- 2) If an error is returned, then the data is too large to fit on the clipboard. The previous contents of the clipboard buffer is still intact (the type and length were not changed). The application should inform the user that the selected data was too large.
- 3) If no error is returned, call **dm_get_clipboard_info** to get the clipboard buffer pointer.
- 4) If no error is returned, the data may be transferred to the clipboard buffer.

Source is the pointer to the buffer area in the application data segment where the information will be copied from.

Dest is the pointer to the clipboard area returned by **dm_get_clipboard_info**.

NumBytes is the size of the clipboard returned by **dm_get_clipboard_info**.

```
copy_to_clipboard ( Source, Dest, NumBytes )
unsigned char *Source;
unsigned char far *Dest;
int NumBytes;
{
    register unsigned char *s;
    register int n;

    s = Source;
    n = NumBytes;

    for ( ; n != 0, n-- )
        *Dest++ = *s++;
}
```


Writing text with attributes to the clipboard:

The text application utilizes attribute description flags in order to display text with the underline or **bold** attributes. These flags must always be used in pairs, with the "ON" flag appearing at the start of the attributed text and the "OFF" flag appearing at the end of the attributed text. Each "ON" flag MUST be accompanied by an "OFF" flag. The following attribute pairs are recognized:

```
UNDERLINE_ON(0x11)   UNDERLINE_OFF(0x10)
BOLD_ON(0x13)        BOLD_OFF(0x12)
```

The defines for these attribute pairs are in the include file, CSRVID.H.

See `form_copy_element` when copying form data to the clipboard.

Writing a 40 Column Application

```
main()
{
    int      video_info, nbr_drivers, driver_index;
    int      i, bFound;
    char      tmp_buff[20];
```

Forty column applications must bind to the CSR using the `dmcsr_bind_init` call which does not load the video driver.

```
/* Bind to the Core Services Resource */
if ( dmcsr_bind_init() == CSR_ERROR )
/* failure to bind to the CSR, could not find/load resource */
    exit(1);

/* Determine number and names of possible drivers */
video_info = vid_loadable_drivers( &VidSwapBuffer[0] );
nbr_drivers = video_info & NBR_DRIVERS_MASK;
driver_index = video_info >> 8;

/* Find the forty column video driver to use on this machine */
for ( i = 0, bFound = FALSE; (i < nbr_drivers) && (!bFound); i++ )
{
    if ( strcmp( VidSwapBuffer[i].driver_name, "DMVST256" ) == 0 )
/* Forty-column video driver for EGA/VGA matches */
        bFound = TRUE;
}

if ( !bFound )
{
    for ( i = 0; (i < nbr_drivers) && (!bFound); i++ )
    {
        if ( strcmp( VidSwapBuffer[i].driver_name, "DMVSTC40" ) == 0 )
/* Forty-column video driver for Tandy 1000 matches */
            bFound = TRUE;
    }

    if ( !bFound )
    {
        for ( i = 0; (i < nbr_drivers) && (!bFound); i++ )
        {
            if ( strcmp( VidSwapBuffer[i].driver_name, "DMVSLRES" ) == 0 )
/* Forty-column video driver for CGA matches */
                bFound = TRUE;
        }
    }
/* did not find a match for Tandy 1000 either, try CGA */
}
/* did not find a match for EGA/VGA try Tandy 1000 or CGA */

if ( bFound )
    driver_index = i - 1;
else
{
    /* did not find a forty column driver - exit application */
    dmcsr_bind_end();
    exit(0);
}
```

The `csr_load_video_driver` actually loads the video driver.

```
ReturnCode = csr_load_video_driver(VidSwapBuffer[driver_index].driver_name);
if ( ReturnCode == CSR_ERROR )
{
    /* could not load the forty column driver - exit application */
    dmcsr_bind_end();
    exit(0);
}

/* Bind with the rest of the core */
csr_access_init();

/* Draw the main screen */
FortyCol_Draw_Screen(VidSwapBuffer[driver_index].driver_name);
```


Note: This example assumes the small memory model.

Writing a DeskMate Resource

A DeskMate resource must provide application and resource side bindings. The application side bindings are linked with the application. The bindings in the DeskMate libraries are application side bindings. These bindings setup the far call into the resource from the application. The resource side bindings are part of the resource. They supply the entry point into the resource and handle returning to the application.

The sample source provided here is included in the SAMPLES\RESOURCE directory.

```
-----  
; APPSIDE.ASM these are the applications side of bindings.  
-----  
include ressegs.inc  
include dmexec.inc          ; include defines for desk  
  
public Resource_Load        ;load the resource  
public Resource_End         ;will remove resource from memory  
public rsc_srqv_vector      ;application binding  
  
_DATA segment
```

The `rsc_srqv` or resource service request vector is the far address of the resource assigned by the executive (desk) once the executive has loaded the resource. This variable is the bridge between the application and the resource.

```
public rsc_srqv  
  
rsc_srqv dw 0abcdh          ;vector to resource binding  
         dw 0dcbah          ;received from desk  
myname   db 'MESSAGE',0     ;name of resource file, without extension  
  
_DATA ends  
_TEXT segment
```


At this point, the application side of the bindings must create a series of jump routines to the `rsc_srqv` address, each routine having a unique function within the resource itself. A macro is used to generate the function jump table. Each time a function call is added or deleted, the jump table must be regenerated.

```

;-----
; Create a jump routines to rec_srq_vector routine with the appropriate value
; in AX. For example:
;
; name of resource app called label near
;     mov ax, name of resource app called
;     jmp near ptr [rsc_srq_vector]
;-----

```

RESOURCE_APP=0

The file `msgbind.inc` contains the macros needed to build application side and resource side bindings for all functions in the resource. Here it is building the application side bindings.

```

ifndef RESOURCE_APP
    asm_proc macro svc_name
        public &svc_name
        &svc_name label near
        mov ax, svc_id
        jmp near ptr [rsc_srq_vector]
        svc_id=svc_id+1
    endm
endif

public functbl

    functbl label word                ;label for beginning of jump table
    svc_id=0
    ;asm_proc goes here

    asm_proc Resource_Message
    asm_proc Resource_Draw_Box

```

This is the actual entry point into the resource.

```

;-----
; rsc_srq_vector will call the resource binding supervisor.
; Entry: ax resource service function.
; Exit: None.
;-----

rsc_srq_vector proc near
    push bp
    mov bp, sp

    add bp, 4                ;get past return address
                           ;ss:bp will point to parameters
                           ; and ax contains function code

                           ;make a service request to resource binding
                           ;on return AX will be set to 0->error, 1->okay

    call dword ptr DGROUP:[rsc_srqv] ;make service request
    pop bp
    ret
rsc_srq_vector endp

```


The resource side of the bindings for this example are in RESSIDE.ASM. The resource side bindings set up the resource information structure defined in RESINFO.INC.

```

;-----
; RESSIDE.ASM these are the resource side of bindings.
;-----
include ressegs.inc

public resource
public Resource Binding
public resource_end

_DATA segment

extrn _ParameterString:byte

.xlist

include dmexec.inc ;defines for desk executive calls
include resinfo.inc ;Resinfo structure and macros
.list

RESOURCE_RES=0 ;build jump table of resources
Set_End _TEXT:resource_end ;use macro to set up EndAddr in struct

;-----
; Set_ResInfo fills in name and binding entry point.
; name must be upper case, no extensions.
;-----

Set_ResInfo RI,"MESSAGE",_TEXT:Resource_Binding

_DATA ends

```

The resource side of the bindings MUST contain a two byte stack for the return address.

```

STACK SEGMENT
    db 2 dup(?)
STACK ENDS

Assume cs:_TEXT, DS:DGROUP

_TEXT segment
include msgbind.inc

```

The resource's stack is only used when the resource is first loaded. After that, the application's stack is used. Also, 2 bytes is not really enough; the interrupt to register the resource will push 4 bytes.

The file msgbind.inc contains the macros which will bind the resource side bindings.

```

ifndef RESOURCE_RES
    asm_proc macro svc_name
        svc_name=svc_id
        extrn &svc_name:near
        dw offset _TEXT:&svc_name
        svc_id=svc_id+1
    endm
endif

public functbl

    functbl label word ;label for begining of jump table
    svc_id=0
    ;asm_proc goes here

    asm_proc Resource_Message
    asm_proc Resource_Draw_Box

```


Besides setting up function numbers for resource calls and providing a near jump into the resource code, the resource side of the bindings saves the current PSP, sets up DS to equal DGROUP, moves the _TEXT segment into CX (for debugging purposes), and asks the executive to execute the resource, and terminate and stay resident for subsequent calls to the resource.

```

;-----;
;_resource saves the current PSP.
; moves DGROUP to the current data segment.
; register the resource with DESK.EXE.
; terminate and stay resident.
;-----;
; Entry:
; Exit:
;-----;
_resource proc near
    mov     cx,es             ;save for PSP
    mov     ax,DGROUP        ;only do this is resource is not in C
    mov     ds,ax
    ;register resource with DESK.EXE
    mov     bx,offset DGROUP:RI ;es:bx =resource info structure
    mov     [bx].ResPSP,cx
    mov     ax,ds
    mov     es,ax
    mov     cx,_TEXT          ;when debugging, use cx to resolve
                                ; _TEXT segment (other segments will
                                ; be offset from there)
    mov     ax,DM_EXEC_RES_START ;interrupt service number
    int     DM_EXEC_INT        ;desk will return ROM/LIM page or ff
;get memory size in paragraphs for TSR
    mov     dx,_LAST
    sub     dx,_TEXT
    add     dx,I0h             ;allow for PSP
    mov     ax,3100h          ;stay resident with DX paragraphs
    int     21h
_resource endp

```


The Resource_Binding address is used by the macro Set_ResInfo as a resource binding entry point for all calls into the resource. Since all calls in the resource enter the resource binding code, this is where the copying of parameters used by the resource functions is done. The function number in AX is doubled and the near call to CS:[BX] results in the actual function being called.

```

-----
; Resource_Binding save all current registers.
; Entry: SS:BP points to first parameter on applications stack.
;        AX    contains function code
;        DS    points to applications data area
; Exit: None.
-----

```

Resource_Binding proc far

assume ds:nothing, ss:nothing, es:nothing

push ax
push bx
push cx
push dx
push si
push di
push bp

push es
push ds ;save segment registers

call copy_params

mov bx,DGROUP
mov ds,bx

shl ax,1
mov bx,0000 ← error: Should be mov bx, functbl
add bx,ax

call cs:[bx]

pop ds
pop es
pop bp
pop di
pop si
pop dx
pop cx
pop bx
pop ax

ret

Resource_Binding endp

Note that all resource functions return void in this example (AX and BX are restored to the caller's values).

This routine copies any parameters needed by the functions. It checks to see which function is being called and copies the amount of data needed by the call. It assumes that DS has been set to the application's data segment.

```

        assume ds:DGROUP, ss:nothing, es:nothing
;-----
; copy_params copies parameters to local data area.
; Entry: SS:BP points to parameters.
;        DS applications data segment.
;        AX function number.
; Exit:  None.
;-----
copy_params proc near
        cmp     ax,Resource_Draw_Box
        jz      no_params
        params:
                ;need to copy parameters,structures to a local data area
                push    ax
                mov     ax,DGROUP
                mov     es,ax

                mov     si,[bp+0]
                mov     di,offset DGROUP:_ParameterString
                mov     cx,0028

                rep     movsb
                pop     ax
                ret
        no_params:
                ret
copy_params endp

```

Note: This resource is not reentrant (parameters copied to static buffer).

The "resource_end" address in the resource side of the bindings is a far address to set the end of the resource's memory in the macro Set_End so that the executive knows which procedure to call before it unloads the resource. This routine should clean up all files, buffers, and close all resources before returning. All latched interrupts should also be unlatched in this procedure.

The sample resource has no open files, resources, or buffer management to take care of, so it simply returns. It is important for the routine to exist for the sake of Desk's code that releases a resource.

```

;-----
; resource_end this routine should clean up all files, buffers and close all
; resources before returning. This is called by desk before
; it is about to remove the resource from memory. All latched
; interrupts should be unlatched at this time.
; Entry: SS:BP points to parameters.
;        DS applications data segment.
;        AX function number.
; Exit:  None.
;-----
resource_end proc far
        ret
resource_end endp

_TEXT ends
end _resource

```


In the example resource MESSAGE.RES, the procedure Resource_Load is the function that requests that the executive load a resource. This call is comparable to the initialization routines **csr_init** and **guf_bind_init**.

```

;-----
; _Resource_Load loads a resource into memory.
;
; Entry: es:dx points to name of the resource module to load.
;        es:bx points to long address.
;
; Exit:  desk will put the resource binding entry point into return AX with:
;        AX=1, Success.
;        AX=0, Failure.
;-----

_Resource_Load proc near
    push    es                ; save es
    push    bx                ; save bx

    push    ds                ; make es = ds
    pop     es

    mov     dx, offset dgroup:myname ; get resource binding name
    mov     bx, offset dgroup:rsc_srqv ; resource entry point will be
                                        ; saved at rsc sqrv
    mov     ax, DM_EXEC_LOAD_RES      ; interrupt service number
    int     DM_EXEC_INT              ; call desk

    pop     bx                ; restore registers
    pop     es

    or      ax, ax             ; test for succes or failure
    jz      error
    mov     ax, 01

error:
    ret
_Resource_Load endp

```

The sample resource MESSAGE.RES also includes an example of freeing a resource, **Resource_End**. This call is equivalent to **csr_end** or **guf_bind_end**.

```

;-----
; _Resource_End calls DESK.EXE to free a resource.
;
; Entry: es:dx name of binding resource.
;
; Exit:  none.
;-----

_Resource_End proc near
    mov     ax, ds
    mov     es, ax            ; make es = ds
    mov     dx, offset DGROUP:myname ; ascii name of resource
    mov     ax, DM_EXEC_FREE_RES    ; free the resource
    int     DM_EXEC_INT

;----- load a dummy routine in the event someone tries to access the resource
;        that we just freed

    mov     ax, offset TEXT:resource_not_loaded
    mov     rsc_srqv, ax
    mov     rsc_srqv+2, cs
    ret
_Resource_End endp

```



```
resource_not_loaded dummy routine to return an error if the resource is  
already gone.
```

```
Entry: None.
```

```
Exit: None.
```

```
resource_not_loaded proc far  
    mov     ax,DM_ERROR  
    ret  
resource_not_loaded endp
```

```
_TEXT ends
```

```
end
```


This code resides in the resource and provides the resource's functionality.

```

;-----
; MESSAGE.ASM - contains two routines, one to display the passed message on
; the screen, the second to draw a box around the displayed
; message.
;-----

include ressegs.inc

.xlist

include csrbase.inc
include csrvid.inc

.list

_DATA      SEGMENT
public _ParameterString
_ParameterString db 28 dup (?) ; Local buffer for passed string
_DATA      ENDS

_TEXT      SEGMENT

extrn _csr_access_init:near
extrn _csr_init:near
extrn _csr_end:near
extrn _vid_clear_screen:near
extrn _vid_move_cursor:near
extrn _vid_set_line_attr:near
extrn _vid_draw_rect:near
extrn _vid_put_string:near

;-----
; Resource_Message - moves the cursor to X = 26; Y = 12.
; displays the message passed at that location.
;
; Entry: None.
;
; Exit: None.
;
; Calls DeskMate's vid_move_cursor, and vid_put_string routines.
;-----

public _Resource_Message

_Resource_Message proc near

    call _csr_init

    mov     ax,12 * CHAR_YEXT
    push    ax
    mov     ax,26 * CHAR_XEXT
    push    ax
    call    _vid_move_cursor
    add     sp,04

    mov     ax,offset DGROUP:_ParameterString ;resource_msg
    push    ax
    call    _vid_put_string
    add     sp,02

    call    _csr_end

    ret
_Resource_Message endp
```



```

; Resource_Draw_Box - sets DeskMate's line attribute to LINE SOLID.
;                      sets DeskMate's line width to LINE WIDTH1.
;                      sets DeskMate's color attribute to COLOR3.
;                      draws a box around the message passed earlier.
;
; Entry: None.
;
; Exit: None.
;
; Calls DeskMate's vid_set_line_attr, and vid_put_string routines.
;

```

```

public _Resource_Draw_Box

```

```

_Resource_Draw_Box proc near

```

```

    call    _csr_init

    mov     ax,COLOR3
    push    ax
    mov     ax,LINE_WIDTH1
    push    ax
    mov     ax,LINE_SOLID
    push    ax
    call    vid_set_line_attr
    add     sp,06

    mov     ax,VID_NO_FILL
    push    ax
    mov     ax,14 * CHAR_YEXT
    push    ax
    mov     ax,55 * CHAR_XEXT
    push    ax
    mov     ax,11 * CHAR_YEXT
    push    ax
    mov     ax,25 * CHAR_XEXT
    push    ax
    call    vid_draw_rect
    add     sp,10

    call    _csr_end

```

```

    ret

```

```

_Resource_Draw_Box endp

```

```

TEXT ENDS

```

```

END

```

```

;-----
; RESSEGS.INC these are the resource segment declarations.
;-----
;

```

```

TEXT segment byte public 'CODE'      ;regular code segment
TEXT ends
PDATA segment byte public 'PDATA'    ;static data
PDATA ends
ITEXT segment byte public 'ICODE'    ;impure code segment (for ROM/LIM)
ITEXT ends
DATA segment byte public 'DATA'      ;regular data segment
DATA ends
BSS segment byte public 'BSS'        ;uninitialized data segment
BSS ends
STACK segment word stack 'STACK'      ;resource stack segment
STACK ends
LAST segment byte public 'LAST'      ; start of data heap
LAST ends

```

```

DGROUP group DATA, BSS, STACK
assume CS:_TEXT, DS:DGROUP, SS:STACK

```


Writing a DeskMate Accessory

DeskMate accessories are mini-applications which perform a specific task for the user. These programs pop-up over the current application and require an application launch them with the **dm_acc_run** function call. There are two types of accessories, the first are application specific and are launched from an application menu option. The second are system accessories which are general purpose and are launched from the F10 menu.

Application specific accessories are called by name. Use accessories to implement functions which are not often used and to reduce the load size of your application. For example, the DeskMate Calendar application uses an accessory to perform its Calendar Merge function. The Calendar application is smaller in size since part of its functionality resides in another program.

System accessories are available in all applications. Developers may write accessories which can be accessed through the More menu option. For more information about "installing" the accessories with the More option, see the **dmmore_add_accessory** function in the Library Functions section of the DeskMate Technical Reference.

Accessories may call **csr_access_init** and **csr_access_end** instead of **csr_init** and **csr_end** if they are NOT going to have a menu bar. Accessories which call **csr_access_init** should call **win_group_init** (and **win_group_end**) to create a new base window. A child window which encompasses the pop-up area should also be created to simplify the accessory's event handling. See the Managing Multiple Windows and Events section for more information.

General Guidelines

- 1) Accessories should be kept small in load size to improve their chances of being launched by any application.
- 2) Accessories should follow the style guidelines for pop-ups discussed in the DeskMate Style Guide.
- 3) Accessories which use resources should:
 - a) check the product version number and on a 3.0 system they should "grow" to 32K to ensure that they are using all available code shed space. Resources cannot be loaded into the code shed space on a 3.0 system. This only applies to resources other than the CSR and GUF which are already loaded on a 3.0 system.
 - b) load the resources as temporary resources using **dm_temp_resource** so that the resources are loaded in the code shed space along with the accessory. This only applies to resources which are not required after the accessory terminates. Accessories which provide functions which require that the resource stay loaded after the accessory terminates should not use this call to load the resource.
 - c) use the resource information in the DESKHDR.EXE utility to inform the executive of what resources are required for execution. The executive will preload the resources and if there is not enough memory for the accessory and the resources it will inform the user.
 - d) use the version number functionality in the DESKHDR.EXE utility to set the file's version number.

Accessory Chaining

Chaining is the method by which two accessories invoke each other, having only one accessory resident in memory at a time. The following code is an example of how an accessory chains to DMHELP.ACC.

Check_ACC_Info is used to save and restore MY_ACC to the state it was in when it chained to DMHELP.ACC. This function writes out the necessary information into the environment manager's environment area before it chains to the help accessory. When the help accessory chains back to MY_ACC, **check_acc_info** is immediately called to restore variables, structures and any needed data to restore MY_ACC. This is done so the user is not aware that they left the accessory.

```
main()
{
    ** bind to necessary resources **

    /* Restore to previous state before help, if necessary */
    Check_ACC_Info(RESTORE);

    /* get input and process until CANCEL */
    do
    {
        event_read(&Event);

        switch(Event.msg)
        {
            case EVENT APPL:
                /* run Help */
                if (Event.param == APPL_ACCESS && Event.x == ACC_HELP)
                {
                    Run_Help();
                    Event.param = ID_CANCEL;
                }
                break;

            case EVENT OUTSIDE:
                *** process outside events **
                break;

            case EVENT CHAR:
                *** process character events **
                break;
        } /* switch */
    } while (Event.param != ID_CANCEL);

    ** unbind to resources **
    return();
}
```



```

int Run_help()
{
    ENVDATA EnvData;          /* environment data structure */
    int Data[40];             /* data buffer */
    register int ReturnCode;   /* return code*/

    /* setup the environment structure */
    EnvData.pEnvFileName = "ACCCHAIN";
    EnvData.pDosEnvString = "ACCCHAIN";
    EnvData.bSwap = ENV_NO_CREATE;
    EnvData.pDmEnvString = "ACCNAME";

    /* see if the environment file exists */
    ReturnCode = env_get(&EnvData);

    /* if the data was not there, create it */
    if (ReturnCode == DM_ERROR)
        env_open(&EnvData);

    Data[0] = ACC_BY_NAME;
    strcpy(&Data[1], "MY ACC");
    EnvData.pDataInfo = (char far *)Data;
    EnvData.DataLen = strlen(Data);

    /* write the environment data to memory */
    env_replace(&EnvData);

    /* Save the necessary information to restore MY ACC to its */
    /* current state when dmhelp.acc chains back */
    Check_ACC_Info(SAVE);

    /* run the help accessory */
    dm_acc_run(ACC_BY_NAME, "DMHELP");
    return(0);
}

```


Part 3
Tools and Utilities

"Tools and Utilities"

Contents

Menu bar Builder - MENUBLD.PDM	3-1
Dialog Box Builder - DLGBUILD.PDM	3-3
Bitmap Editor - HYPERBIT.PDM	3-7
Graphics Form Generator - DRAWLIST.PDM	3-9
Clipart File Builder - CLIPART.PDM	3-11
Stroke Font Editor - STROKE.PDM	3-13
Memory Map Generator - MEMMAP.PDM	3-15
Desk Header - DESKHDR.EXE	3-17
Disk Label Generator - DMLABEL.PDM	3-21
Customized Runtime Utility - RUNTMLD.PDM	3-25
Customized INSTALL.EXE Utility - INSTLBLD.PDM	3-26

Must find/replace all occurrences of `_MENU_TAG` with `_APP_TAG` in .h file created by this utility. Otherwise, the code will not compile.

Menu bar Builder - MENUBLD.PDM

The Menu Bar Builder allows the programmer to interactively build an application's menu bar. The lines of source code declarations and definitions necessary to describe the application's menus are generated automatically for the programmer. A 'C' language header or include file (.H), ready for use in the programmer's application, is produced.

The Menu Bar Builder is a DeskMate application which follows the DeskMate Style Guide, providing both a mouse and a keyboard interface. The information needed to describe the application menus is entered in two dialog boxes.

The information which describes the menu bar - menus, buttons, help, message menu, and accessories menu - is entered in the "Menubar" dialog box. The menu titles for the F2 through F8 menus are entered in the designated edit fields. The Alarm Menu, Tandy Accessories Menu, Help, and Up, Down, Left, and Right arrow buttons are selected through their respective check boxes.

The information which describes the individual menus is entered in the "Items Definitions" dialog box. The Item edit field is used to enter each item's string. The Accelerators list box is used to assign an accelerator to an item. Check boxes are used to *enable* or *check* a menu item. Radio buttons are used to set the menu item's *group*. The list box to the right displays the current menu's name and any items already defined. The ADD push button is used to add another item to the menu. The DELETE push button is used to remove an item from the menu.

This utility makes the creation of a DeskMate application's menu bar quick and easy. A complete menu bar definition can be created in a matter of minutes. The programmer need not worry about the contents of the data structures, the constant definitions necessary, or the dimensions of the menus while creating the menu bar.

Source code definitions and declarations generated include initialized string declarations, return code definitions, individual MENUITEM and MENU data structure declarations, and the MENUBAR data structure declarations. The code generated by the utility is of a generic nature. You may choose to change the naming conventions used before incorporating the code into your application. The utility is meant to create the initial menu bar definition, simple changes to the menu bar definition can be made manually by editing the header file.

Summary of Commands:

File Menu :

New	Creates a new menu bar definition. Prompts the user to save any changes first.
Open...	Opens an existing menu bar definition header file.
Save	Saves the menu bar definition to the header file.
Save as...	Prompts the user for the name of the header file, generates and saves the menu bar definition to named file.
Exit	Exits the utility.

Run...

Standard File Run function, prompts the user for the name of the application to be executed when the utility exits.

Options Menu :

Define menubar...

Displays the "Menubar" dialog box.

Define items...

Displays the "Items Definitions" dialog box.

Dialog Box Builder - DLGBUILD.PDM

The Dialog Box Builder provides the programmer with a way to interactively design and execute dialog boxes. The true power of the program becomes evident when the lines of source code declarations and definitions necessary to describe the dialog box are generated automatically. The 'C' language header or include file created is ready for use in an application. The routine to actually invoke and interface with the dialog box is the piece of code written by the programmer. By saving the *binary image* of the dialog box, the programmer can come back and make changes, regenerating the header file as needed.

Note: the builder will fail if you try to put more than 40 components in the box. There are also limits on the number of each type of component. The limits are cumulative: each time the box is edited, you add more components toward the limit, but components deleted do not count toward increasing the limit. The max number of static strings is 14.

The Dialog Box Builder is a DeskMate application which follows the DeskMate Style Guide, providing both a mouse and a keyboard interface. Components are added by positioning the text cursor (block) first, then choosing the component to add. The Components Menu contains the commands to add the individual components. Dialog boxes are used to enter the component information such as size or dimension, data type, and label.

The standard user interface components are supported - push buttons, radio button groups, list boxes, check boxes, edit fields, and a menu bar. Static strings are provided for labels and prompts. Static boxes are also provided to show component groupings.

This utility makes the development of DeskMate applications quick and easy. A complete dialog box, including the title, frame, and components can be created in a matter of minutes. The programmer need not worry about the contents of the data structures while designing the dialog box. The utility automatically calculates the coordinates necessary to center the box on the screen. Changing the tabbing sequence and selecting components for modification, repositioning, or removal are also supported.

The source code generated includes initialized string declarations, edit field buffer declarations, edit field format string declarations, return code definitions, individual component data structure declarations, the redraw and component array data structure declarations, and the frame and dialog box data structure declarations.

Summary of Commands:

File Menu :

New	Creates a new dialog box, prompts the user to save changes first.
Open...	Opens an existing dialog box file (.dlg), prompts the user to save changes first. Clears screen, displays the dialog box.
Save	Saves the dialog box image to file.
Save as...	Saves the dialog box image to a new file for editing purposes later.
Exit	Exits the utility.
Run...	Standard File Run function, prompts the user for the name of the application to be executed when the utility exits.

Source Menu :

Generate...	Creates the source header file for the dialog box. Prompts user for a label prefix to be used when generating variable declarations.
Modify variables...	Displays dialog box used to cycle through the components, allowing the user to change the default names of the source variables.
Extension length...	Prompts the user to enter the number of characters of the dialog box name to be used in generating the names for the source variables. This feature is useful when several dialog boxes are being created.

Components Menu :

Pushbutton	Adds a push button component at the current cursor location. Brings up the Push Button Definition dialog box, allowing the user to enter the component specific information. Displays the push button when the user OKs the box.
Radiobutton	Adds a radio button group component at the current cursor location. Brings up the Radio Button Group Definition dialog box, allowing the user to enter the component specific information. Displays the radio button group when the user OKs the box.
Listbox	Adds a list box component at the current cursor location. Brings up the List Box Definition dialog box, allowing the user to enter the component specific information. Displays the list box when the user OKs the box.
Checkbox	Adds a check box component at the current cursor location. Brings up the Check Box Definition dialog box, allowing the user to enter the component specific information. Displays the check box when the user OKs the box.
Editfield	Adds an edit field component at the current cursor location. Brings up the Edit Field Definition dialog box, allowing the user to enter the component specific information. Displays the edit field when the user OKs the box.
Menubar	Adds a menu bar component at the current cursor location. Brings up the Menu Bar Definition dialog boxes, allowing the user to enter the component specific information.

Static string Adds a static string at the current cursor location. Brings up the Static String Definition dialog box, allowing the user to enter the text string. Displays the string when the user OKs the box.

Static box Adds a static box at the current cursor location. Displays a default size box at the cursor location, with a size handle in the lower right-hand side of the box. The user drags the handle to resize the box.

Modify Modifies the currently selected component or static string. Brings up the Definition dialog box, allowing the user to change the component / string specific information. Displays the component / string when the user OKs the box. When the currently selected item is a static box, the sizing handle appears, allowing the user to resize the box.

Delete Removes the currently selected item from the dialog box.

Options Menu :

Title... Allows the user to enter or change the dialog box title.

Width+Height Repositions the lower, right-hand side of the dialog box to the current cursor location - sets the new width and height of the dialog box.

Tab Order Allows the user to change the component tab order for the dialog box.

Location Box... Brings up a dialog box that allows the user to select the corner of the screen where the X and Y location box appears. The location box contains the origin of the selected component. This box is used to align components in a dialog box.

Run dialog Allows the user to "try out" the dialog box.

Redraw screen Redraws the screen.

Bitmap Editor - HYPERBIT.PDM

The Bitmap Editor provides the programmer with a way to interactively modify device-independent bitmaps. The bitmaps may be created from Draw data files (.fig). These bitmaps may be used in information boxes or simply displayed on the screen.

The Bitmap Editor is a DeskMate application which follows the DeskMate Style Guide, providing both a mouse and a limited keyboard interface. The information needed to describe the device-independent bitmap is generated in several formats - as a graphics form in a Draw file format (.fig), as a graphics form 'C' data structure definition header or include file (.h), and as a graphics form assembly language data structure definition header or include file (.inc).

Hyperbit supports all DeskMate video modes, including the Tandy 4 and 16 color modes.

This utility makes the modification of device-independent bitmaps simple by allowing full scrolling while in the *zoomed editing* mode. A select rectangle is used to determine the area of the bitmap to initially zoom in on. Double-clicking on the bitmap will toggle the zoom/normal modes. Keyboard accelerators of Ctrl+Z to zoom and Esc to toggle back are also supported.

The size of the bitmap in bytes is displayed on the main screen at all times, notice the size change when the Bits per pixel is changed. Two, four, and sixteen color bitmaps are supported. The more colors used, the larger the bitmap.

Summary of Commands:

File Menu :

New	Creates a new bitmap file, prompting the user to save changes first.
Open...	Opens an existing bitmap file (.fig, .h, or .inc), prompts the user to save changes first. Clears screen, displays the new bitmap.
Save	Saves the bitmap information to file (.fig, .h, or .inc).
Save as...	Saves the bitmap information as a graphics form, which may be read by Draw, or as a 'C' or assembly language header file.
Exit	Exits the utility.
Run Draw	Runs the Draw application automatically after exiting.

Edit Menu :

Copy	Places a copy of the bitmap on the clipboard as a graphics form which may be pasted into the Draw application for further manipulation.
Paste	Pastes in a graphics form from the Draw application, converts it to a device-independent bitmap for editing.

Zoom	Zooms in on the bitmap, allowing editing of the actual pixels.
Options Menu :	
Bits per pixel...	Displays dialog box allowing the user to choose to make the bitmap a two (1 bit), four (2 bits), or sixteen (4 bits) color bitmap.
Change colors...	Displays a dialog box used to change one color of the bitmap to another. For example, in a four color bitmap all the "green" pixels can be changed to "red".
Clear background	Toggled check mark menu item used to set the bitmap background to transparent. When displayed, graphics under the bitmap will show through the background.
Solid background	Toggled check mark menu item used to set the bitmap background to solid. When displayed, graphics under the bitmap will be covered by the bitmap background.

Graphics Form Generator - DRAWLIST.PDM

The Graphics Form Generator converts a graphics form which was placed on the Clipboard by Draw (or other application which uses CLIP_DRAW data type) to its equivalent 'C' source header or include file. This utility allows a programmer to create a screen or picture using the Draw application and then compile into an application, making it part of the program's data.

A simple DeskMate application, Draw List, automatically pastes in a picture from the clipboard at start-up and prompts the user to paste after a task switch.

Use the video call **vid_draw_form((FORM_HDR *) Form, x, y)** to display the bitmap at the desired location on the screen.

Summary of Commands:

File Menu :

Save as...	Saves the graphics form as a 'C' source header file.
Exit	Exits the utility.
Run...	Standard File Run function, prompts the user for the name of the application to be executed when the utility exits.

Clipart File Builder - CLIPART.PDM

The Clipart File Builder allows the software developer to create clipart files with the extension CLP from Draw files with the extension FIG. Clipart files can also be created by copying objects to the clipboard from Draw and pasting them into the Clipart application. Clipart files that are to be released to the public should not contain blank pictures. The following are the three methods of operation used to build clipart data files.

Method 1:

1. Create one or more pictures which will be included in a clipart file using DRAW.PDM.
2. Select each picture and make it a separate object (use the Make Object menu option).
3. Save your pictures to a disk file. Only a draw file that contains all objects can be made into a clipart file.
4. Run CLIPART.PDM.
5. Open the draw file from the clipart application.
6. Select a display layout from the Display menu - 2 objects, 3 objects, to 12 objects, depending on the number of pictures in your file.
7. Rearrange the individual pictures using the clipboard.
8. Save the pictures as a clipart file.

Method 2:

1. Create pictures using DRAW.PDM. Select the pictures you wish to be made into clipart and make each one a separate object.
2. Copy each picture to the clipboard. Only clipboard pictures that are objects can be made into clipart.
3. Run the clipart application, CLIPART.PDM.
4. Select a display layout from the Display menu - 2 objects, 3 objects, to 12 objects, depending on the number of pictures in your file.
5. Paste the pictures into the clipart application.
6. Rearrange the individual pictures using the clipboard.
7. Save the pictures as a clipart file.

Merging Files:

1. Open a file and use "Copy all" from the "Edit" menu.
2. Open a second file and use "Paste" from the "Edit" menu.
3. Select a larger display layout to display the merged pictures.

Summary of Commands:

File Menu :

New	Creates a new clipart file. The user is prompted to save changes first.
Open draw file...	Opens a .fig file created with Draw. The file contains only one object.
Open clipart file...	Open a .clp clipart file.

Save draw file as...	Saves the data as a Draw file with a .fig extension.
Save clipart file as...	Saves the data as a Clipart file with a .clip extension.
Exit	Exits the utility.
Run...	Standard File Run function, prompts the user for the name of the application to be executed when the utility exits.

Edit Menu :

Cut	Cuts the selected clipart to the clipboard.
Copy	Copies the selected clipart to the clipboard.
Paste	Pastes in the contents of the clipboard.
Clear	Deletes the selected clipart.
Copy all	Copies all displayed clipart to the clipboard.

Display Menu :

Allows the user to select the number of objects to be displayed from 2, 3, 6 (horizontally and vertically), and 12.

Stroke Font Editor - STROKE.PDM

Provides the programmer with a way to edit or create his own stroke font definitions to add characters to an alphabet or modify an existing alphabet. Stroke definitions may be created for icon buttons, such as those used to display the tools in the Draw application.

Stroke is a DeskMate application which provides only a mouse interface. The screen is divided into two windows, one for editing of the font definition and one for displaying the result at any size.

A Draw figure may be used to "trace" over while creating a font definition. An optional grid may also be used to help position lines. The point value is displayed on the screen.

A "stroke" is defined as a pen down, draw line, draw line, ..., pen up. In the draw mode, a mouse click or double-click designates a pen up/down. Press and drag is used to draw lines within a stroke.

In select mode, a mouse click is used to select a line segment for resizing, moving, and deleting.

Summary of Commands:

File Menu :

New	Creates a new stroke font definition file, prompts the user to save changes first. Clears screen.
Open...	Opens an existing stroke font definition file (.h), prompts the user to save changes first. Clears screen, displays the last character definition.
Save	Saves the font information to file.
Exit	Exits the utility.
Run...	Standard File Run command, runs the specified program when the utility exits.

Edit Menu :

Copy	Places a copy of the character definitions as a text string in a graphics form.
Paste stroke	Pastes in a graphics form which contains a text element which uses a Draw font - such as the Roman font alphabet.
Paste overlay	Pastes in a graphics form from Draw which is used to "trace over" during the editing or creation of a new definition. Most useful for the creation of icon pictures.
Delete line	Deletes the selected line.

Delete stroke

Deletes the group of lines defining the stroke the current line is a part of.

Character Menu :

Next

Displays the next character in the definition, always enabled since there is always a next "blank or new" character in the list.

Previous

Displays the previous character in the definition. This item is disabled when viewing the first character.

Delete

Removes the current character definition.

Rename

Renames the character being defined.

Select

Selects a character definition, versus using the next and previous commands to cycle through all the characters.

Display all

Displays the full character set defined in the current file.

Options Menu :

Select brush

Displays the Brush Styles dialog box, uses the selected brush when displaying the font.

Draw mode

This mode allows the user to add new lines to the character definition. This is a checked menu item with Select mode item which follows.

Modify mode

This mode allows the user to move, resize, and delete lines in the character definition. This is a checked menu item with the Draw mode item above.

Redraw screen

Redisplays the screen.

Overlay

This mode displays the overlay. This is a checked menu item, which is toggled on/off.

Grid

This mode displays a grid the user may use for alignment during drawing. This is a checked menu item, which is toggled on/off.

Memory Map Generator - MEMMAP.PDM

One of the most commonly asked questions is "How much system memory does DeskMate use?". The Memory Map Generator answers that question, given any system configuration.

The Memory Map Generator is a small DeskMate application which scans the memory arenas determining the allocated and free segments and their sizes (in bytes). The information is displayed in a list box on the screen. The information is gathered once, upon execution of the program. The utility does not appear as an arena entry, it takes itself out of the calculations.

To determine the amount of space available for an application, run MEMMAP .PDM.

To determine the amount of space available for another application to task switch with your application, first run your application then task switch to MEMMAP .PDM.

To determine the amount of space left for an accessory, rename MEMMAP .PDM to DMHELP .ACC and execute the by using the F1 key. Remember to save a copy of the Help Accessory before copying over the Memory Map utility.

Summary of Commands:

File Menu :

Save as	Saves the memory map information as a Text ASCII document, for editing or printing purposes.
Exit	Exits the utility.

The Desk Header Utility - DESKHDR.EXE

The Desk Header Utility reads an .EXE file and its associated MAP file and creates a DeskMate executable file, .PDM, with the DeskMate extended header. The extended header contains information about the file for the executive's use in loading and executing the program. You should run your application through the utility each time it is relinked. See the make file distributed with the Samples for more examples of the utility's use. DESKHDR.EXE can add the following items to the header:

Shed size This is the number of paragraphs that DeskMate may code shed in order to run an accessory. This number currently is only used for .PDM files.

Split address DeskMate may load a program in two separate, non-contiguous pieces if there is a split address in the header. This makes it more likely that a program might fit if memory is fragmented and there is not a single chunk of memory large enough to contain the program. If a program has a split address in its header, the executive will load the first part of the program (up to the split address) in one piece and load the second part of the program (all the rest from split address up) in another piece. The PSP will be immediately before the second part.

extra memory This is where the program may notify the loader that it intends to allocate extra memory during execution. The current version of the executive does not use this information.

resources If an accessory will require one or more (up to 5) resources to be loaded in order for it to run it can declare these in the header so that the executive may load these resources before loading the accessory. This way the executive can find out that it needs to code shed the current application in order for the accessory and all its resources to fit.

min & max heap These are the numbers that were used by the SETHEAP .EXE program. These values optimize the amount of memory required before the executive can load the program. Each value is a number of paragraphs.

The values for MIN and MAX are added to the existing minalloc value, which the linker placed in the file header, to get the new values for minalloc and maxalloc.

When the program does not plan to allocate any memory on its heap, the values for MIN and MAX should both be zero (0).

When a program wants the maximum amount of heap space a small- or middle-model C program can use, the word "full" is used instead of a numeric value. The utility will attempt to open the MAP file for the program to determine where the data segment DGROUP starts and calculates the heap to extend to 64k past the beginning of DGROUP. The DESKHDR utility will not run without a map file.

C programs use part of their heap space to store the arguments (argv) and the environment when the program begins execution. The linker does not account for this space in setting the initial value for minalloc. If your C program needs to use heap space, you should add an amount to the MIN and MAX values to allow for storing the arguments and environment. Even if the program does not use any heap, you may want to add some heap to allow for these values. If C does not have enough heap for these strings during start up, it shrinks the stack by the amount of memory required for them.

- version** A version number may be stored in the extended header. This number can be used to quickly track versions of a program by always having one place to look for the version of any DeskMate program file. The executive does not currently use this information.
- lim flag** A flag in the header tells the Desk Executive that the first code segment(s) or specified code segment(s) of the program may be loaded into LIM expanded memory. Only supported in the Runtime and Retail (stand-alone) products.
- shadow ram flag** A flag in the header tells the Desk Executive that the program may be loaded into shadow RAM. Only supported in the Runtime and Retail (stand-alone) products.

To run DESKHDR.EXE you must create a control file which specifies the names of the output file and the input files and the contents of the extended header fields. To run the utility type

deskhdr <control>

where <control> is the name of the control file.

Control file format

keyword	argument	default
=====	=====	=====
Output	<filename>	none
Input	<filename>	output filename with .exe ext
MAP	<filename>	input filename with .map ext
VERsion	<symbol name> or <number>	_VERSION_NUMBER
SHED	<Segment name> or <number>	_ITEXT, or DGROU if no _ITEXT
SPLIT	<Segment name>	_ITEXT
MEMory	<number>	0
Resource	<resource name>	none
MINheap	<number>	0
MAXheap	<number> or "Full"	full
LIM	<Segment name> or <none>	if not specified do NOT load into LIM
SHADow	<none>	if not specified do NOT load in shadow RAM

The keywords may be abbreviated using the letters shown in caps. The commands may appear in any order.

If VERSION specifies a symbol name, the symbol is the address of a word in the program containing the version number of the program.

<number>s are in paragraphs (16 bytes each) unless followed by "k", in which case the number is assumed to be in k-bytes (1024 bytes each). Numbers may be given in decimal, octal, or hex. Hex numbers have "0x" before first digit. Octal numbers have 0 for first digit. Decimal numbers should not start with 0.

MIN heap and MAX heap will be added to the minalloc value in the input file to compute the minalloc and maxalloc values for the output file. If maxalloc of the input file is not 0xFFFF, the heap calculations will be skipped and minalloc and maxalloc will be left unchanged. MAX heap full means DGROUP will be 64k total.

Resource is used for accessories to tell desk which resource(s) it will need. There may be up to 5 Resource lines per file. The <resource name> should not include the .RES extension.

Output is the only required command line in the control file.

If the LIM keyword does not exist in the control file, then the DeskMate application or accessory will not be put into LIM. The segment name parameter for the LIM keyword is the LIM boundary. If a parameter is not specified then the Desk Header program will attempt to locate an _FTEXT segment in your application or accessory .MAP file. If an _FTEXT segment is not found then the Desk Header program will search for an _ITEXT segment. If either segment is found, the specified application or accessory will be placed in LIM from the beginning of the specified application or accessory to the found segment. If a segment name is specified on the LIM command line, the Desk Header program will put the address of that segment in the header in the place of the _FTEXT and _ITEXT segments. If the specified segment name is not found or if the LIM keyword is used and _FTEXT or _ITEXT segments cannot be found, an error message is displayed by the Desk Header program.

If the SHADOW keyword does not exist in the control file, then the DeskMate application will not be put into shadow RAM.

Examples

Example 1 - Minimum file requirement:

```
O MYAPP.PDM
```

This control file will cause the DESKHDR program to use MYAPP.EXE and MYAPP.MAP as input to create the file MYAPP.PDM.

The program's code shed size will be the size of its _TEXT segment. (Assuming _ITEXT or DGROUP follows _TEXT in map.)

If it has an _ITEXT segment it will be loadable in two separate pieces, _TEXT in one piece, and its PSP, _ITEXT, DGROUP, STACK, and its heap in the other piece.

Its minimum heap requirement will be 0 and its maximum heap will be everything from the end of the program (above STACK) to 64k past the beginning of DGROUP.

If there is a symbol `_VERSION_NUMBER` in `MYAPP.MAP`, the word at that address will be copied into the version number field of the header.

`MYAPP.PDM` will not be loaded into LIM or shadow RAM.

Example 2 - Uses resources and sets the heap sizes:

```
O MYACC.ACC
R DMDb
MIN 0x100
MAX 15k
```

This control file will cause the DESKHDR program to use `MYACC.EXE` and `MYACC.MAP` as input to create the file `MYACC.ACC`.

If it has an `_ITEXT` segment it will be loadable in two separate pieces, `_TEXT` in one piece, and its PSP, `_ITEXT`, `DGROUP`, `STACK`, and its heap in the other piece.

The minimum memory required to run `MYACC.ACC` will be the size of the program including all its initialized and uninitialized data plus its stack plus 256 paragraphs (1024 bytes). The maximum amount of memory that will be allocated to the program, and the amount of memory `desk.exe` will attempt to free up for the program will be the size of the program including all its initialized and uninitialized data plus its stack plus 15 times 1024 bytes.

`MYACC.ACC` will not be loaded into LIM or shadow RAM.

Example 3 - Uses LIM:

```
Output MYAPP.PDM
MIN 10
MAX FULL
LIM C_ETEXT
```

This control file will cause the DESKHDR program to use `MYAPP.EXE` and `MYAPP.MAP` as input to create the file `MYAPP.ACC`.

Since the `LIM` keyword is used, the LIM flag will be set in the header for `MYAPP.PDM`. The Desk executive will place the first part of the program into LIM. The size of the LIM segment is determined by the LIM boundary, for this example `C_ETEXT`.

Disk Label Generator - DMLABEL.PDM

The Disk Label Generator allows the software developer to create a reference file that DeskMate will use to prompt a user for when disk swapping. The reference file, LABEL.LBL (required name), contains a list of application and data filenames which are associated with the name of the disk containing those files. When your application requires a particular file and DeskMate cannot find that file on the current disk in the drive, DeskMate will access the file LABEL.LBL and copy its contents into the DeskMate environment managed by the Environment Manager in the GUF resource. It will retrieve the disk name from the DeskMate environment and ask the user to insert by name, that disk you designate as the home of the file. On subsequent attempts to access a particular file that cannot be found on the current disk, DeskMate will retrieve the disk name from the copy of LABEL.LBL that resides in the DeskMate environment.

The utility also lets you store information in the reference file that can be used by the install program, INSTALL.PDM, used to copy files from floppies to a hard disk system. Eight bits of information is stored for each file. The following is a list of the current designation of each bit:

Bit 0 (LSB)	Never copy
Bit 1	Copy on date compare
Bit 2	Copy if Deskmate 3.0 CSR
Bit 3	Copy to DMCONFIG directory
Bit 4	Copy if older version
Bit 5	not currently used
Bit 6	not currently used
Bit 7 (MSB)	not currently used

Creating a label file:

1. Manually create a master set of disks for your product.
2. Run DMLABEL.PDM.
3. Put the first disk in drive A or B and select "Add new disk..." from the "Options" menu. The disk will be read for all files in the root directory and a dialog box will be displayed which lists all files in the "Files" list box.
4. At the "Disk label:" prompt, type the name of the disk with which you wish to prompt the user when trying to access any file on this disk.
5. If there are any files on the disk for which you do not wish the user to be prompted by disk name, select such files in the "Files" list box and push the "DELETE FILES" push button. The LABEL.LBL file and application data and configuration files should not be included.
6. If there are any files on the disk for which you wish to set any bits of the installation flags, select such files in the "Files" list box and place an "X" in the appropriate check box. The default selection is to always copy the file.
7. Push the "ADD NEW DISK" push button to accept all data in the dialog box and enter it into memory.

8. The contents of the added disk will appear on the screen. Standard DeskMate cursor keys (Up arrow, Down arrow, Page Up, Page Down, Ctrl+Home, Ctrl+End) may be used to view the entire contents of data in memory.
9. To add the next disk, go to step 3 and substitute "next" for "first". Continue until all disks have been processed.
10. To save the data in memory to a file on disk, select "Save as..." from the File menu. For the released product you should name the file LABEL.LBL (this is the only label file name that will be accessed by DeskMate when looking for a file). However, it is possible to save the file using different names if you will have different versions of the LABEL.LBL file produced for different products (e.g. you may have one version for 3 1/2" disks and one for 5 1/4" disks). The file LABEL.LBL should reside on each disk of the released set.

Changing a disk label:

1. Select "Change disk label..." from the "Options" menu.
2. At the "Disk Labels" list box, select the label you wish to change and push the CHANGE push button.
3. A second dialog box will appear. At the "Change to:" prompt, type the new disk label.
4. Push the "CHANGE" push button to change the disk label.

Deleting a disk of files:

1. Select "Delete disk..." from the "Options" menu.
2. At the "Disk Labels" list box, select the label of the disk you wish to delete.
3. Push the "DELETE" push button to remove the label and all files associated with this disk label. The disk number of any disks following the deleted disk will be adjusted accordingly.

Adding files to a disk:

1. Select "Add files..." from the "Options" menu.
2. Put the disk containing the files to be added in drive A or B. The disk will be read for all files in the root directory and they will be displayed in a list box after choosing the disk label.
3. At the "Disk Labels" list box, select the label of the disk to which you wish to add one or more files and push the "OK" push button.
4. If there are any files on the disk which already exist in the label file or for which you do not wish the user to be prompted by disk name, select such files in the "Files" list box and push the "DELETE FILES" push button.

5. If there are any files on the disk for which you wish to set any bits of the installation flags, select such files in the "Files" list box and place an "X" in the appropriate check box.
6. Push the "ADD FILES" push button to add all files in the "Files" list box to the previously selected disk. Duplicate filenames are not allowed, you will be informed of any duplications.

Changing file(s) installation flag bits:

1. Select "Change files..." from the "Options" menu.
2. At the "Disk Labels" list box, select the label of the disk which contains the file(s) for which you wish to change installation flag bits.
3. Select any file(s) from the "Files" list box and place an "X" in the appropriate check box.
4. Push the "OK" push button to accept all changes.

Deleting files from a disk:

1. Select "Delete files..." from the "Options" menu.
2. At the "Disk Labels" list box, select the label of the disk which contains the file(s) you wish to delete.
3. At the "Files" list box, select the file(s) you wish to delete.
4. Push the "DELETE" push button to remove all selected files.

Moving files from one disk to another:

1. Select "Move files..." from the "Options" menu.
2. At the "Disk Labels" list box, select the label of the disk which contains the file(s) you wish to move.
3. At the "Files" list box, select the file(s) you wish to move.
4. At the "Move To" list box, select the label of the disk to which you wish to move the file(s).
5. Push the "Move" push button to move all selected files.

Label file format:

The label file is a DeskMate environment file.

Each filename has a key field for which there are two bytes of data :
byte one for disk number
byte two for installation flags.

All files for disk #1 are listed in the label file before all the files for disk #2, which are listed before all files for disk #3, etc.

Following all filenames is the LABELS key field for which there are several bytes of data which includes each label string for each disk in order of disk number. The use of the word, LABELS, as a key field prevents any files in your label file from being named LABELS.

Each key field is null terminated. The two bytes after the null terminator are length bytes which determine how many bytes of data that follow are associated with the key field. For key fields that are filenames, the length is always 2. For the LABELS key field the length is dependent on the total length of all label names.

... In other words, all the variables in LABEL.LBL are filenames, each with 2 bytes of associated data. The first byte is the disk number, starting at 1; the second byte is the flags listed on page 3-21. The last variable, called LABELS, has a variable amount of associated data, consisting of the diskette labels, null-terminated, in order starting from 1. A very simple format.

Customized Runtime Utility - RUNTMBLD.PDM

The Customized Runtime Utility allows the programmer to create a customized 3.05 executive which displays your company's copyright message and automatically executes the specified application instead of the DeskMate DeskTop application. This utility replaces the DeskMate 3.2 BLDRUNTM.PDM utility which is still used to customized 3.2 executives. These utilities are NOT interchangeable and you must match the utility to the executive being customized.

A dialog box is used to enter the following information to customize your runtime executable:

- Applications name: APPLNAME (8 characters, extension will be .PDM)
- Customized name: APPLNAME (8 characters, extension will be .EXE)
- Copyright message: 40 characters.
- Forty/Eighty column driver choice.
- Lowest video resolution flag.

The executive file RUNTIME.EXE file is loaded, modified in memory, and the customized file, APPLNAME.EXE is written out. To insure the correct version of RUNTIME.EXE is used, copy the file into the current directory. The new file will be created in the current directory.

The Forty/Eighty column driver choice tells the CSR which set of drivers to auto-detect from. The Lowest video resolution flag tells the CSR to choose the "worst-fit" video driver when it auto-detects the video. For instance, most videos would auto-detect CGA except where the video is Hercules.

This utility MUST be used with and ONLY with a DeskMate 3.03 or later runtime file. To customize a DeskMate 3.2 runtime, use the BLDRUNTM.PDM distributed with the DeskMate Development System 03.02.00.

Customized Installation Launcher Utility - INSTLBLD.PDM

This utility allows the developer to create a small `INSTALL.EXE` program used to install a DeskMate application in a stand-alone environment. This program uses the application's customized runtime executive to launch the `INSTALL.PDM` application at installation time. This utility enables a developer to use the same customized runtime on the diskette to launch both the application and the installation application.

The utility displays a dialog box which prompts the user for two pieces of information:

- 1) The name of the customized runtime without the `.EXE` extension, for example, `MYAPP` for `MYAPP.EXE`.
- 2) The **Disk Label** used in `DMLABEL.PDM` for the diskette containing the customized runtime file and the `INSTALL.PDM` program.

This utility needs the file `INSTALL.TEM` and your customized runtime, `MYAPP.EXE`, to reside in the current directory with the utility. The utility will open both files, patch `INSTALL.TEM` in memory and write out `INSTALL.EXE`.

Refer to *Distributing Your Application* for more information about the installation procedure.

Part 4
Distributing Your Application

"Distributing Your Application"

Contents

The DeskMate Checklist	4-1
Installation and Upgrade Procedures	4-3
Determining DeskMate Product Versions	4-5
Installation launched from the DeskTop	4-5
Installation launched from the INSTALL.EXE	4-5
How to get the file version	4-6
Runtime Distribution Guidelines	4-7

The DeskMate Checklist

Programs that will be sold by Radio Shack as DeskMate applications must meet these requirements:

1. The program must be implemented using the DeskMate Development System and use the DeskMate environment.
2. The program must be installable using the DeskTop's F7 Menu, Install option. Refer to the Installation and Upgrade Procedures section which follows this Checklist for more information.
3. The program must support the DeskMate 3.2 and use the DeskMate version 3.3 or later help system.
4. The program should run all accessories (including "More...") and have the F10 menu button on its menu bar.
5. The program must permit task switching from the F10 menu.
6. If the program uses a cut/copy/paste function, the program should support the DeskMate clipboard as its cut/copy/paste buffer. If the program has graphics capabilities, it should use the DeskMate Forms Manager to permit the data to be transferred in the DeskMate graphics format.
7. The program must have the F9 notification menu button enabled.
8. If the program changes the user-defined colors, the program must restore the colors to those specified by the user when the program terminates.
9. The program must not use DOS overlays. If new portions of code must be overlaid onto an executing program, the program should use DeskMate Resources instead of overlays.
10. The program should use the DeskMate printer drivers.
11. The product must be submitted to Radio Shack Computer Merchandising for interface and style guide approval before the application can bear the trademarked DeskMate User Interface logo.
12. The product package should display the trademarked DeskMate User Interface logo.

Installation and Upgrade Procedures

All DeskMate applications should have an installation program which is itself a DeskMate application. The installation program should be easy to use and not alter the user's system without the user being notified.

The installation program should not perform DOS commands which might alter the user system (other than creating directories and copying files), such as setting the date and time, deleting AUTOEXEC.BAT or CONFIG.SYS files, or modifying AUTOEXEC.BAT or CONFIG.SYS files such that the user cannot easily recover.

Whenever appropriate, the user should be given a choice to continue the process or cancel. For instance, if the installation is about to delete all system files from a previous version of your product, the program should inform the user giving the option to approve or cancel the process.

Every application must:

Have an INSTALL.EXE program which launches the application's INSTALL.PDM file from a DeskMate 3.05 runtime. This program is used to install stand-alone versions of a product. Use the INSTLBLD.PDM program to build your customized INSTALL.EXE program. This file must be on the same diskette as the application's customized version of RUNTIME.EXE.

Have an INSTALL.PDM application which copies files to the user's hard disk using the following guidelines:

Create a directory for the user in which the files are installed. Present the user with a default pathname which can be modified.

When installing on a DeskMate product, do not copy the DeskMate system files unless an upgrade has been recommended or your product requires the version of your runtime. The installation program should determine the DeskMate version by the method outlined in the Determining the DeskMate Product Version section which follows.

To install as a stand-alone system, copy the DeskMate system files along with your application's files to the directory.

If your product uses the DeskMate Help system, copy the application help file to the directory along with the application.

Neither the INSTALL.PDM nor INSTALL.EXE files should be copied to the hard disk.

For 40 column applications, the INSTALL.PDM must also be a 40 column application.

Do not install the DeskMate system files in a directory which contains a DeskMate product or another vendor's runtime, unless Tandy has recommended you upgrade a system due to an incompatibility or to fix a known problem. A runtime installation should never downgrade/upgrade a user's DeskMate system as it might inadvertently cause the system to no longer function properly. Check the version number of a file before upgrading the user to ensure the user's

NOTE: Do not install the 3.03 help manager and 3.03/3.02 help compatibility utility in the main DeskMate directory under 3.02 or before. If you want to provide ~~text~~ ~~the~~ help under pre-3.03 systems, you must install in a subdirectory. Otherwise, if you install in the main directory, you cannot provide ~~the~~ ~~your~~ help in DM 3.02 & before. See runtime documentation.

product is not mistakenly downgraded.

Provide a DESKTOPD.CFG configuration file which is used by the Desktop install function. This file should be copied along with the other application files during the installation.

Create this file using the DeskTop Menu (F7) Create Quick Load option.

DESKTOPD.CFG is used by the QUICK LOAD application box on the DeskTop. When the user changes directories to your directory, the box will change to show your application and list of data files.

Have a diskette label file, LABEL.LBL, created with DMLABEL.PDM which contains diskette information used in file searching and for diskette prompts. The file also contains instruction flags for each file which tell the installation program how to copy the file. The diskettes must also have unique volume id's used by your customized INSTALL.EXE and the file search function to prompt for diskettes. The diskette label file should not be copied to the hard disk.

To provide help during the installation process, an INSTALL.HLP help file can be supplied with the product. This file must reside with INSTALL.PDM and should not be copied to the hard disk. If you choose not to provide help during the installation process then this file is not needed.

The user documentation for installation on a DeskMate DeskTop should give the following directions:

- 1) The user should be told to insert the diskette (use the name from the label program) which contains INSTALL.PDM into any floppy drive.
- 2) Direct the user to use the Desktop Menu (F7) Install option to install your application on the DeskTop.
- 3) The user should then follow the prompts given by your installation program.

To reinstall or upgrade on a DeskMate 3.2 or later system, the user should be instructed to use the Desktop Menu (F7) Delete option to remove the application's definition and then follow the installation directions outlined above.

The user documentation for installation or upgrade of a stand-alone system should direct the user to:

- 1) The user should be told to insert the diskette (use the name from the label program) which contains INSTALL.EXE into any floppy drive.
- 2) Direct the user to change to that drive.
- 3) The user should then run INSTALL.EXE to do the installation.

If your application allows the user to make backups of the product diskettes, then the user should be directed to use the DISKCOPY command to insure the volume id's are copied when the diskettes are copied.

Determining DeskMate Product Versions

Installation launched from the DeskTop

The installation program, INSTALL.PDM, can detect if it was invoked from the DeskTop through the F7 Install option by calling `env_open` with the following ENVDATA structure.

```
ENVDATA your_env =  
{  
    USER.CFG,  
    DMCONFIG,  
    ENV_NO_CREATE,  
    USER,  
    (char far *)0,  
    0,  
};
```

*This does not work.
`env_open()` never returns
DM_ERROR.*

If `env_open` does not return DM_ERROR, then install was invoked from the DeskTop.

If invoked from the DeskTop, do the following to determine the DeskMate version:

```
ret_code = dm_inquire_product();  
if ((ret_code & DM_VERSION) != 0)  
    user has DeskMate 3.3 or greater  
else  
    user has DeskMate 3.2 or less
```

Before copying the necessary files (based on the DeskMate version) to a directory you must make sure the DeskMate product is not in that directory. If none of these files are found, then DeskMate is not in the directory.

- 1) Ensure DESK.EXE is not present.
- 2) If it is not present, then check for a Tandy ROM machine in which the file is in ROM. Check that at least three of the following files are not in the directory, since it is possible that one of these applications may be in ROM:

```
ADDRESS.PDM  
CALENDAR.PDM  
FORMSET.PDM  
FILER.PDM  
DRAW.PDM  
TEXT.PDM
```

If your runtime executive and application files are present, you can consider this installation to be an upgrade and copy the files.

If your executive and application files are not present, see if any DeskMate 3.0 runtime resource, .RRS extension, or your runtime files are present. If so, there is another runtime application in that directory and you should not install in this directory.

Installation launched from the INSTALL.EXE

If INSTALL.EXE invoked INSTALL.PDM, you have to search the system to determine if

DeskMate is present.

```
ret_code = dm_file_search("DESK.EXE", pPathbuffer, 0);
if (ret_code == 1)
    The file was found and the path is in pPathbuffer
else
    The file was not found, so call dm_file_search for the
    DeskMate application files listed above.
```

If none of these files are found, the user does not have the DeskMate product.

How to get the file version

As long as your application, accessory, and resource files use the DESKHDR.EXE utility, you can determine the version of your files in the manner described below. Do the following to determine the version of the file:

- 1) Open the file, refer to this FileHeader structure for variable offsets.

```
struct FileHeader    =
{
    int      MagicBytes[12];
    int      RelocSeek;
    int      VersionNum;
    char      DM89Key [4];
};
```

- 2) The element RelocSeek must be greater than 25H.
- 3) The element DM89Key must contain the four bytes "DM89".

If items 2 and 3 are met, then the DeskMate file is version 3.3 or greater. This method can be used by your INSTALL.PDM for upgrading only files with prior versions. The element VersionNum contains the DeskMate 3.3 (or greater) version number. The format of this element is file dependent, for DeskMate resource files, *.RES, the version number is binary. DeskMate application and accessory files use ASCII version numbers.

Runtime Distribution Guidelines

Only distribute files listed in Exhibit A of the DeskMate Distribution License. Files marked for non-distribution should not be distributed.

Do not distribute mixed versions of the DeskMate system files. For instance, do not distribute the 3.2 versions of any of the accessories with the 3.3 resources or vice versa.

Files which MUST be distributed with your product:

RUNTIME.EXE	Executive - Distribute your customized version
INSTALL.TEM	Runtime Installation Launcher - Distribute your customized INSTALL.EXE version. You must write the INSTALL.PDM program which is launched by this program.
INSTALL.PDM	Your DeskMate application which installs your application onto a hard disk.
DMSETUP.ACC	Setup Accessory
DMSETUP.HLP	Setup Accessory Help File
DMCSR.R89	Core Services Resource
PRGUF.RES	Power & Run General User Functions Resource
DMMDJ.RES	Tandy 1000 Joystick Driver (DMMDJOY.RES)
DMMDP.RES	Micro-Channel Serial Mouse Driver
DMMDS.RES	Serial Mouse Driver (DMMDSERT.RES)
DMSSM.RES	Screen Saver Resource
DMEMM.RES	Extended Memory Manager Resource
DMVID.EXE	DeskMate video force utility.
DMVID.DOC	Video force utility documentation.

Distribute the video driver resolution set which is required by your application. If your application is a standard 80 column application, distribute ONLY these drivers:

DMVS1000.RES	Tandy 1000 (TGA), 4 color video driver
DMVSCGA.RES	CGA, 2 color video driver
DMVSEGA.RES	EGA, 16 color video driver
DMVSHERC.RES	Hercules, 2 color video driver
DMVSVGA.RES	VGA, 16 color video driver
DMVSTC16.RES	Tandy TL/SL (ETGA), 16 color video driver
DMVST.RES	EGA board, CGA monitor, 2 color, 640 X 200 resolution video driver
DMVSMCGA.RES	MCGA, 2 color video driver

If your application is a 40 column application, distribute ONLY these drivers:

DMVSLRES.RES	40 column, low resolution video driver
DMVST256.RES	40 column, vga video driver
DMVSTC40.RES	40 column, Tandy 1000/TL/SL video driver
DMVSH.RES	40 column, Hercules video driver

DMVSE.RES	40 column, EGA video driver
DMVSM.RES	40 column, Monochrome EGA video driver

Files which must be distributed ONLY if your applications uses the specific function or resource:

DMPGSET.ACC	DeskMate Page Setup Accessory
DMPGSET.HLP	DeskMate Page Setup Accessory Help File
DMHELP.ACC	Help Accessory
DMHELP88.ACC	DeskMate 3.0 Compatible Help Accessory
DMHLPENG.RES	DeskMate Intelligent Help Resource
DMGUF.R89	General User Functions Resource
DMDB.R89	Database Control Resource, is required by the DeskMate Help System
DMDBBLD.RES	Database File Build Resource (DBBUILD.RES)
DMDBRD.RES	Database File Read Resource, is required by DeskMate Help System (DBREAD.RES)
DMDBUPD.RES	Database File Update Resource (DBUPDATE.RES)
DMFORM.RES	Form Manager Resource
DMEFORM.RES	Extended Form Manager Resource
DMFONT.RES	Font Resource
DMTHES.RES	Thesaurus resource (see local dealer).
DMPDASCI.RES	Daisy-wheel, or other non-supported printer, printer driver
DMPDIBMM.RES	IBM-compatible graphics printer driver
DMPD1.RES	Tandy DMP 105 printer driver (Tandy mode)
DMPD2.RES	Tandy DMP 200, 420, or 430 printer driver (Tandy mode)
DMPDLASR.RES	HP Laserjet Plus or Laserjet-compatible printer driver
DMPDS.RES	24-pin extended printer driver
DMPD.CFG	Configuration file where printer information is saved
DMPRTSEL.ACC	General printer selection accessory which makes use of:
DMPRTSEL.HLP	Printer selection accessory help file
DMPDASCI.ACC	Daisy-wheel, or other non-supported printer selection accessory
DMPDIBMM.ACC	IBM-compatible graphics printer selection accessory
DMPD1.ACC	Tandy DMP 105 (Tandy mode) printer selection accessory
DMPD2.ACC	Tandy DMP 200, 420, or 430 (Tandy mode) printer selection accessory
DMPDLASR.ACC	HP Laserjet Plus or Laserjet-compatible printer selection accessory
DMPDS.ACC	24-pin extended printer selection accessory
PLAY.PDM	Play application, launches tutorial or demo
DMPLAY.RES	Play resource
DMUNPACK.RES	Tutorial Decompression Resource
DEMO.PDM	Customized Runtime Demo Launcher
TUTKBD.RES	Keyboard Layout Resource

If your application makes use of the font technology include these video and printer drivers in addition to the regular video and printer drivers:

DMVE1000.RES	Tandy 1000 (TGA), 4 color video driver
--------------	--

DMVECGA.RES	CGA, 2 color video driver
DMVEEGA.RES	EGA, 16 color video driver
DMVEHERC.RES	Hercules, 2 color video driver
DMVEVGA.RES	VGA, 16 color video driver
DMVETC16.RES	Tandy TL/SL (ETGA), 16 color video driver
DMVET.RES	EGA board, CGA monitor, 2 color, 640 X 200 resolution video driver
DMVEMCGA.RES	MCGA, 2 color video driver

DMPEIBMM.RES	IBM-compatible graphics font printer driver
DMPE1.RES	Tandy DMP 105 font printer driver (Tandy mode)
DMPE2.RES	Tandy DMP 200, 420, or 430 font printer driver (Tandy mode)
DMPELASR.RES	HP Laserjet Plus or Laserjet-compatible font printer driver
DMPES.RES	24-pin extended font printer driver

and the associated resident font definition files needed for the video and printer drivers:

DMPDASCI.RFD	Daisy-wheel, or other non-supported printer, printer driver
DMPDIBMM.RFD	IBM-compatible graphics printer driver
DMPD1.RFD	Tandy DMP 105 printer driver (Tandy mode)
DMPD2.RFD	Tandy DMP 200, 420, or 430 printer driver (Tandy mode)
DMPDLASR.RFD	HP Laserjet Plus or Laserjet-compatible printer driver
DMPDS.RFD	24-pin extended printer driver

and the associated font files

COBB.FF1
DIXON.FF1
MARIN.FF1

Part 5
DeskMate Help Systems

"DeskMate Help Systems"

Contents

Overview	5-1
The Intelligent Help Manager	5-1
The Help Queue	5-1
The Help Accessory	5-1
Help Display	5-1
Writing the Application Help File	5-5
Writing the Help Window Text	5-5
Format	5-5
Help on	5-6
Menus	5-6
Menu options	5-6
Grayed menu options	5-6
Dialog Boxes	5-7
Message Boxes	5-8
Edit Fields	5-8
List Boxes	5-8
Other Components	5-8
Mouse vs. Keyboard	5-9
Writing the Help Screen Text	5-10
Format	5-10
Help Topic	5-10
Numbered help buttons	5-10
Screen Titles	5-10
The first help screen	5-10
Common Problems	5-11
How to Fill Up Screens	5-11
Creating the Sample Help File VIDEO.HLP	5-13
General help	5-13
Help when a menu option is highlighted	5-13
Help in dialog boxes	5-17
Rule-based help in a dialog box	5-18
Help groups	5-21
Help Rule Base Utility - DMHELP.UTL	5-25

Adding menu bar text	5-25
Adding new rules	5-25
Group numbers	5-25
Queue Data	5-26
Functions	5-26
DeskMate Help Editor - DMEDITOR.PDM	5-29
Working with rules	5-29
Editing a rule	5-29
Adding a new rule	5-29
Rules that chain	5-29
Working with displayed text	5-29
Size restrictions	5-29
Link to rules	5-29
Working with solution extensions	5-30
Working with button topics	5-30
Help File Compression Utility - TOKEN.PDM	5-31
Help File Format	5-33

Overview

The Intelligent Help Manager

The Intelligent Help Manager, IHM, is a part of the Core Services Resource. It tracks the user as he uses DeskMate. Each time the user starts up a new application or accessory, the IHM starts gathering information on the context of the application. The IHM is called every time your application runs the application menu bar, calls `dlg_run`, `msg_run`, or `cmp_run`. The IHM determines whether or not to keep any information about the component that is running.

The Help Queue

Because the information is so diversified, the IHM keeps a wrap-around queue of component entries. The IHM is selective about the information it stores. For example, when the user runs a dialog box, the manager adds a dialog box entry to the queue and stores the edit field, list box, radio button, icon button and check box information about the dialog box with it.

The Help Accessory

The Help Accessory takes the queue information and the application help file to find the best help solution for the current context of the application. If no solution is found, general application help is displayed.

The accessory has a built-in *inference engine* that is *rule-based*. It uses a *backward chaining algorithm* to find a help solution. Since you write the *rules* for your application, you have the flexibility of making the help as specific as the IHM's queue information will allow. You also decide in what cases you want help to be displayed.

The inference engine takes the data out of the queue and translates it into *facts*. The rules you define are made up of *premises* and a *conclusion*. The conclusion is TRUE if all the rule's premises are TRUE. When checking if the premises are true, the inference engine compares them to its facts. A match means the premise is true. The premises in a rule do not have to be facts and they can be conclusions for other rules - the start of backward chaining.

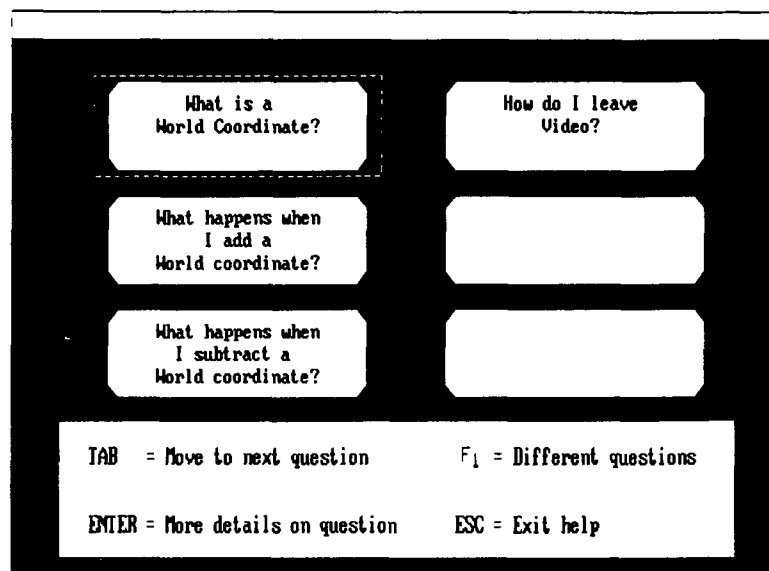
Help Display

A conclusion can be a text solution, a group, or a SuperGroup. If the help you want displayed has multiple step or multiple topics, your conclusion should be a group, `$<groupname>`. For the most general help available, the conclusion should be a SuperGroup, `$$<supergroup>`. The help accessory will display the SuperGroup help if no other solutions are found. There should only be one SuperGroup solution in your help file. The SuperGroup is the first level of *general help* the user will see when no solutions or conclusions could be made for the current context of the application.

A text solution displays *specific help* on an item in a *help window*.



The group and SuperGroup solutions redraw the screen and display six (6) *topic buttons*. The user can choose the help topic they want by selecting one of the buttons. We will allow five levels of help, so that you can go into further detail as you go down a level.



When the help displayed in a help window is not sufficient, you can attach another text solution (another help window) or a group solution (a button screen) to your conclusion. When the user presses F1 in the help window, the attached text or group will appear. Using the help editor, you can specify the "Next solution" for the current text or group solution.

Definitions:

A *Topic* is a single solution with text to display with it.

A *Group* has one or more groups or topics in it. A maximum of 18 groups + topics.

A *Title* is a descriptive string to help the user identify with the contents of a group or topic.

You can define the starting level for General Help to be any group, not just the SuperGroup. You can add rules that will cause a group to be displayed. This means you can have different starting levels depending on your application's current state.

You can use a rule to display a group depending on your state. For example, the DeskTop has a completely grayed Sort By Menu on the menu screen, but these items are available when the tree screen is viewed. In a case like this, you can define 2 groups, \$Sort by and \$Sort2, where \$Sort by defines all the gray options and \$Sort2 defines the enabled options. Normally, an internal rule would return \$Sort by, but an external rule could be defined for \$Sort2 :

```
if
  CALL MenuItemChecked (Tree)
  menu.item.selected (Sort by)
then
  $Sort2
```

There will probably be topics that you want to include with your application that no rule will lead to. These topics will be available whenever running full screen help, help invoked by the application by running the Help Accessory. This *basic help* should be provided for all menus, menu options, dialog boxes, message boxes. In addition, help can be provided for individual components in the work area.

Specific help is help you define with the rules you write yourself. These rules can utilize the queue information, like checking the contents of a dialog box edit field.

You can make rules that take past user activity into account. This will help you solve problems like having dialog boxes with the same name. These dialog boxes will come up in different situations, perhaps one after another. You can take advantage of the fact that multiple events will get you to a unique state when you write your rules.

Writing the Application Help File

Now that you know how the Help system works we should discuss how to actually write your application's help file. First we will cover the style guidelines you should follow when writing the help, then we will demonstrate how the help tools were used to actually create a help file for one of the sample applications, then the details of the help tools will be covered.

Writing the Help Window Text

Format

You should use the same structure for each of your help windows. Use the following guidelines:

First introduce the help topic.

Now describe what the user has to do, or point out errors.

Put any extra information related to this topic at the end.

[END]

Introduce the topic

Make sure the first sentence the user sees will make it clear what the help will be about.

Do not indent.

If you are going to make paragraph transitions, do not indent the first line of the new paragraph.

Do not use blank lines.

Use a line containing ----- to provide a transition between paragraphs. Blank lines do not encourage the user to continue reading the help, especially if the blank line appears at the bottom of the window.

Extra information appears at the end.

Any extra information you think the user might like to know should appear last.

Mark the end of the text.

Every help window should end with [END] to make it clear that there is no more help in this window. The Help Editor does this for you automatically.

Help on...

Menus

Give a general explanation of the functionality of the options on the menu. Refer to the menu by name. Do not link a menu directly to a screen of help, let that transition occur after the first window of help. That is, go ahead and link the general explanation to a help screen. This screen should contain topics that relate to the help window. For example, the help for the DeskTop File Menu is:

```
DeskTop lets you manage the files
that reside on your disk. You can
make copies of files, remove old
files from the disk, or change the
name of files. [END]
```

Menu options

The help for these options should always refer to the option by name, and should explain what the option will allow the user to do. The option name should be capitalized. For example, the DeskTop File Copy option provides the following help:

```
The TYPE option lets you sort the
files by the filename extension. The
extension is the three letters
following the period. [END]
```

You may also use the phrase: "Choosing the <menu option name> option lets ..." . You may also make reference to the Menu on which the option resides. For example, the COPY option on the File Menu

Grayed menu options

The help for these options should always refer to the option by name, and should explain what the option will allow the user to do. The option name should be capitalized. Next, tell the user that the option is grayed and why. The "why" part of your help should state directly, or imply, what the user needs to do to enable the option. Never use the word disabled; always refer to the option as grayed. For example, the DeskTop Type option gives the following help when it is grayed:

```
The TYPE option lets you sort the
files by the filename extension. The
extension is the three letters
following the period.
-----
The TYPE option is grayed because
this function is only available at
the Tree screen. [END]
```


Dialog Boxes

Start dialog box help with an explanation of what task the box will perform. Then outline what the user should do at each prompt, ending with how to complete or abort the operation. If there are a large number of prompts to describe, use more than one window of help by attaching another window (fill in the Next Solution field). This way, the user will not be overwhelmed by a single window full of details. If your dialog box is fairly simple, you may want to skip the explanation of the task and proceed to the steps involved in completing the operation. For example, the DeskTop help for the Format Disk Dialog Box is:

```
Enter the name of the drive in which
you wish to format disks.
```

```
-----
If you have a high-capacity drive,
at Options: type /n:9 /t:80 to
format a 720K 3 1/2 inch disk, or
type /4 to format a 360K 5 1/4 inch
disk.
```

```
-----
Check the install operating system
box if the diskette will be for
startup. [END]
```

An example of more complex dialog box help is the help on the Change Directory Dialog Box:

```
The Change Directory box is used to
change the current drive or
directory (displayed on the title
line of the DeskTop).
```

```
-----
Type in the letter of the drive
followed by a colon, then the name
of the directory.
```

```
For example : A:\NEW
```

```
-----
There must be a disk in the new
drive specified, and the directory
you type must be an existing
directory. Use the TREE option on the
View Menu to see all directories on
a drive. [END]
```

This screen links to another screen that talks about directories. If the user presses F1 he will see:

```
A directory is a collection of files
on a disk much like a folder inside
a file cabinet. In addition to
files, a directory can also contain
other directories.
```

```
-----
The Directory menu lets you look at
the contents of a directory, make
new directories or remove empty
directories. [END]
```


Message Boxes

The help you provide for a message box should be a more detailed explanation of the problem that has occurred, and how to solve it. Try to be as specific as possible. For example, in DeskTop, if you type an invalid directory name you will get a message box stating "Path was not found." The help displayed for this box is:

```
A part of the directory name typed
either contains invalid characters
or could not be found on the disk.
-----
All directories on the disk can be
viewed from the Tree View.
-----
Check the directory you specified
for accuracy, and then press ENTER.
[END]
```

Edit Fields

Explain what you can type into the editfield, and for what purpose. For example the Address Book gives the following help for the Title field in an address record:

```
You can type a courtesy title using
ten characters or fewer. This title
will be printed on your labels.
[END]
```

List Boxes

Indicate what is in the list box, and why the user might want to select an item. Also indicate how one (or more) items can be selected. For example, in Address Book, the help for the Index List Box is:

```
When you select a name from the
Index List Box, the accompanying
address record is displayed.
-----
Press PG UP or PG DOWN to display a
page of names, or press the up or
down arrow keys to move the
highlight one name at a time.
-----
You can also click the mouse button
on a name to select it as the current
address record.
-----
If you are looking for a particular
name, type the first letter of that
name to display the address records
beginning with that letter. [END]
```

Other Components

If you have help for the other components (check boxes, icon buttons, and radio buttons. Use the same philosophy as help on edit fields. Give help on what the component symbolizes, not what it is. There is no need to tell the user that he is looking at an icon

button. For example, The Draw application uses icon buttons to represent tools. The help always refers to them as such.

Mouse vs. Keyboard

When you start giving the user instructions, you will be faced with the dilemma of explaining how to perform a task with either the keyboard or the mouse. We recommend the following:

1. When instructing the user to push a button, tell them the key that they should press, rather than where to click with the mouse. Never give mouse-only instructions.
2. If you feel the help can easily be explained for both the mouse and the keyboard, explain both methods.
3. If the message you are trying to get across will be lost by the explanation of the keyboard and mouse sequence then use a keyboard explanation only.
4. Do not mix keyboard and mouse instructions. Do not tell the user to do one step with the mouse and the next with the keyboard. You may, however, tell them they can do one step by keyboard or mouse, and then just explain the keyboard method for the next step.
5. Use the phrases "double click the mouse button" and "single click the mouse button" when referring to those actions.

Writing the Help Screen Text

Format

Help Topic

Use a question format to identify a help topic. For example, some of the buttons seen in DeskTop help are:

How do I change the DeskTop display?
How do I move around on the DeskTop?
What is a directory?

You should, therefore, use {how, what, when, where} as often as possible. If you cannot describe a topic using a question, then you may use a phrase or a complete sentence. The more specific you are, the better. The user should have a strong idea of what he will see if he selects a button.

Numbered help buttons

Whenever you present the user with a screen of buttons that represent the steps in completing a task, use the numbering capability. The button labels on these screens are not in a question format, they are in an action format. If you do put some general information on this screen (that is not one of the steps), that button should not be numbered. To number buttons in the help editor, choose the "Number this field" check box in the Edit Topic dialog box or, insert an extra star (*) delimiter in FRONT of the first solution you want numbered. The numbering will stop when an extra star (*) delimiter appears AFTER a topic.

Screen Titles

Always use titles on numbered screens. The title should be the same that appeared as the topic of the button the user selected to get the numbered help screen. Since you have more space available, you may use a more descriptive title if you feel it is necessary. To encourage the user to read all the buttons on a help screen, do not use titles on any other screens.

The first help screen

The SuperGroup help screen is the first full help screen the user sees unless you create rules that go to other help screens. The SuperGroup screen is a general overview of the help topics that direct the user to more specific help topics. If you create a rule for another screen, that screen should have topics specific to the application's current state. If six topics are not enough to describe either or these cases, then link this screen to other screens. When the user presses F1, the next set of topics will be displayed.

Common Problems

If there are common problems people experience when using your application, then the user should be able to see these topics by selecting a button labeled:

Common
Problems

Place a screen(s) of help underneath the button.

How to Fill Up Screens

Do not feel obligated to fill up all six buttons on a help screen. Instead, concentrate on grouping topics that relate to one another. (Exception: SuperGroup screen. Since it will contain an overview of your application, is not likely to follow this convention).

One of the help screens you should include in your file is a help screen. It is the last screen the user will see if he continually pressed F1 to view all your screens. This screen is included in the `STARTER.HLP` file, its solution string is `$Help`.

Creating the Sample Help File VIDEO.HLP

To explain the actual process necessary to build an application help file we will show how the help tools were used to create the help file for the sample application VIDEO.PDM. The types of help we will create and edit are:

- General help
- Help when a menu option is highlighted
- Help in dialog boxes
- Rule-based help in a dialog box
- Help groups

The following tools and utilities are provided to aid in the creation and editing of an application help file. These files and tools were used to build the VIDEO.HLP help file for the VIDEO.PDM application.

DMHELP.UTL	Help rule based utility
DMEDITOR.PDM	Help text editor
TOKEN.PDM	Help file compression utility
STARTER.HLP	Basic help file template
STARTER.TKN	Starter help file token file

General help

The file STARTER.HLP contains the help text used in most applications, such as how to use menus, dialog boxes, edit fields and other components. Other help text includes use of the mouse, common file i/o and database error messages, how to use accessories, and other subjects common to many DeskMate applications.

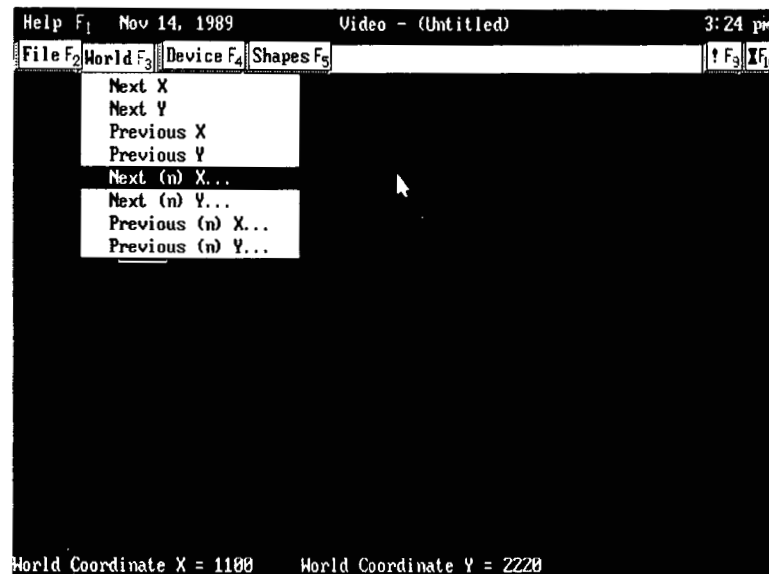
To begin, STARTER.HLP is copied to create the file VIDEO.HLP to provide general DeskMate application help.

Help when a menu option is highlighted

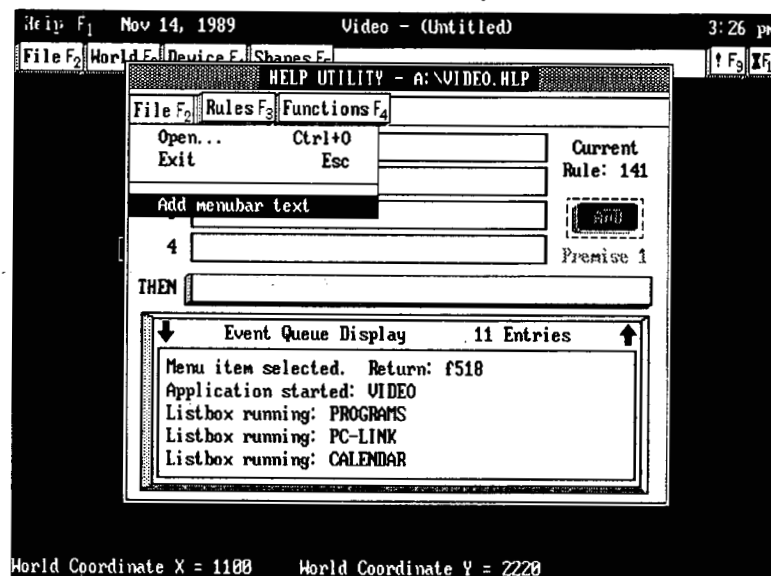
Once the file VIDEO.HLP has been created, it is used as the basis for the application-specific help to be added.

The accessory DMHELP.UTL is renamed DMHELP.ACC. Now, whenever the F1 key is pressed, this utility will run in the place of the Help accessory. Let us look at how help for all of the menu options in VIDEO.PDM was created.

First we will add help for Next (n) X... in the World F3 Menu. The Video application is run and the option in the menu is highlighted.



The F1 key is pressed to invoke the help utility.



The event queue display contains a list of the events that led to the selection of the Next (n) X... menu option. The return code, f518, is the code for the menu option. While in the Help Utility, Add menubar text is selected from the File F2 Menu. Now every return code for every menu option in the Video application will be added to VIDEO.HLP.

To examine how each menu option return code has been added to the help file for the Video application, the Help Editor, DMEDITOR.PDM is used. The data file VIDEO.HLP is selected as the file to edit.

In the Help Editor File F2 Menu, Switch to text is selected to examine the help solutions and text. For the solution string f518, our Next (n) X... menu option, the editor displays

Help F₁ Nov 14, 1989 HelpEdit - A:\VIDEO.HLP 3:33 pm

File F₂ Text F₃ Edit F₄ ! F₉ F₁₀

Solution string: f518

Help text to present: World,Next (n) X... [END]

Next solution:

Notice how the Help Utility inserted the return code for the menu option as the *Solution string* and the actual menu option string as the *Help text to present*.. Every menu option in your menu bar will appear this way. It is now a simple manner to enter the desired help text to replace the string for the menu option.

Help F₁ Nov 14, 1989 HelpEdit - A:\VIDEO.HLP 10:52 am

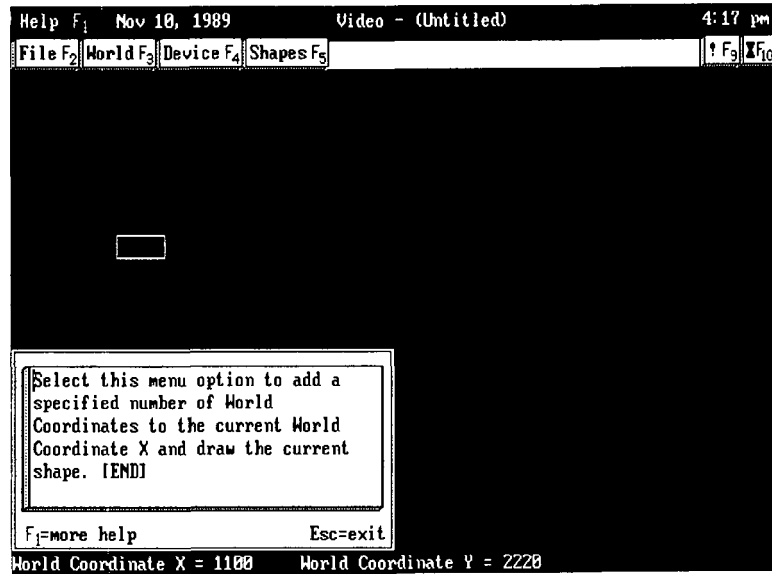
File F₂ Text F₃ Edit F₄ ! F₉ F₁₀

Solution string: f518

Help text to present: Select this menu option to add a specified number of World Coordinates to the current World Coordinate X and draw the current shape. [END]

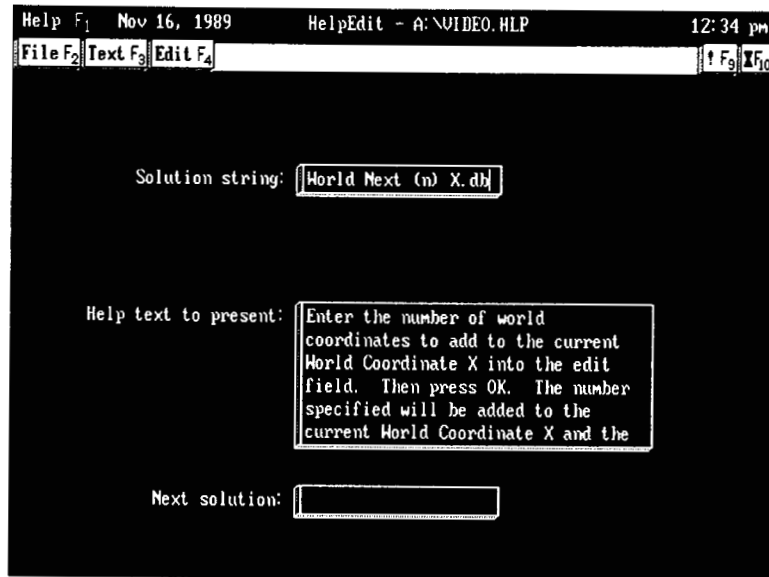
Next solution:

Now after exiting the Help text editor and returning to VIDEO.PDM, when the Next (n) X... option is highlighted in the World Menu and F1 is pressed, the help text entered in the Help text editor will appear.

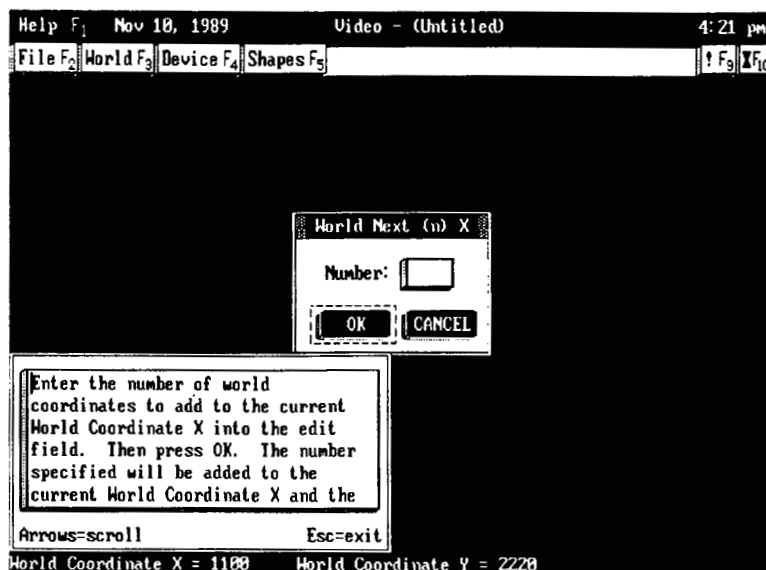


Help in dialog boxes

Setting up a help screen that explains the components in a dialog box or how to enter information in a specific dialog box is simple. Run the help text editor, get into Text mode (CTRL+T) and select New from the Text F3 Menu. Now enter the name of the dialog box,, as it appears in the dialog box frame along with the .db identifier in the Solution string edit field. Then enter whatever text you wish to appear as help for the dialog box.



Add rules or premises with the Help Utility for dialog box help when you want to check the state of a specific component within the dialog box. When F1 is pressed in the dialog box, the help text entered will appear.

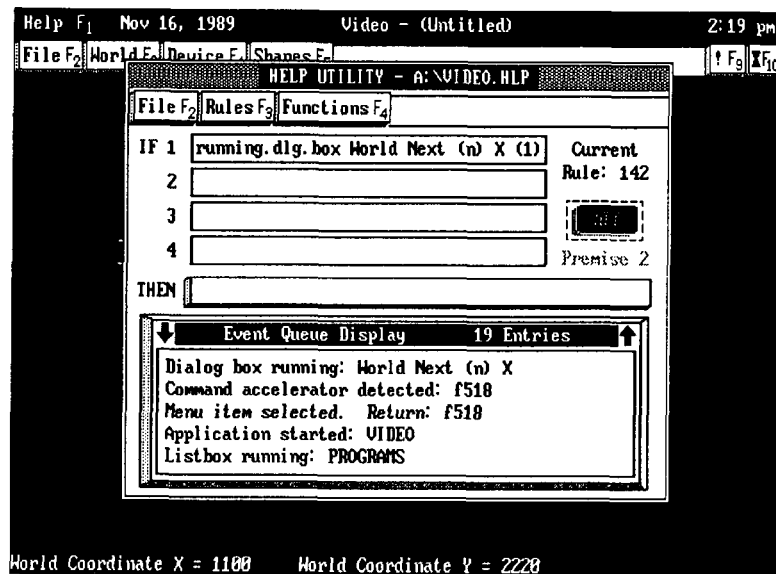


Rule-based help in a dialog box

As an example, we will show how to add help for the World Next (n) X dialog box. First the desired dialog box is brought up on the screen.



When F1 is pressed, the help utility appears with a list of events in the help queue. Arrowing down to the last event (running.dlg.box World Next (n) X (1)), will display the event as the first premise. Pressing enter at this point adds the event as a premise to this rule.

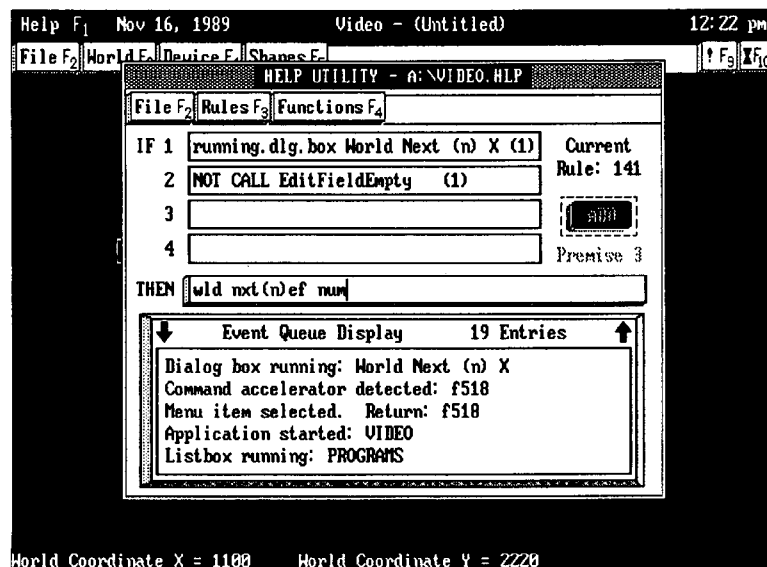


With this premise included, any subsequent checks we wish to make about the state of components in the dialog box can be made. We wish to bring up a certain help screen if the edit field in the dialog box is already displaying a number.

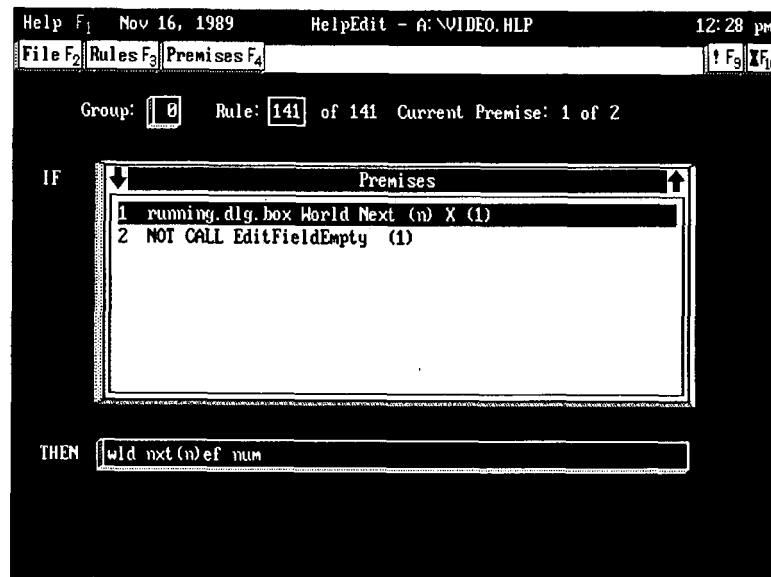
At this point, arrowing down into the first item in the Event Queue Display Listbox will once again place the premise running.dlg.box World Next (n) X (1) in the second premise editfield. Selecting the Call option from the Functions F4 Menu will allow setting the check for the edit field contents.



Pressing the NO button here will mean that the solution will be invoked if the user has pressed F1 while the World Next (n) X dialog box is running AND the edit field has some number displayed. Once NO has been pressed, the solution string (an arbitrary string for use by the Help Editor) is entered. This string will correspond to the actual help text to be displayed.



Now we leave the Help Utility and the application to edit the help text for the rule we just set up. Running the Help Editor in the Rule mode will show the rule and solution string.



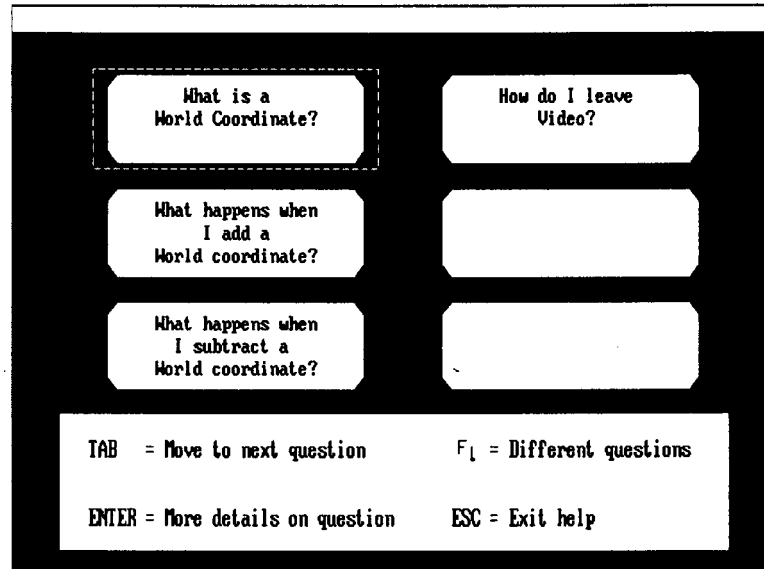
Switching to Text mode and finding the needed solution string "wld nxt(n)ef num" allows us to enter the actual help text that corresponds to the rule we defined.



Since this dialog box already has help text defined which is displayed if the edit field is empty (the case which has no premises), the rule for the more specific case of an edit field with characters displayed should appear first in the help file - it should have a lower rule number.

Help groups

Address help for specific questions with a group of help topics. A help group appears as a group of up to 6 beveled rectangles, each with a unique subject and solution.



To enter a help group, we will once again use the "Next (n) X" menu option. When this option is selected by the user and F1 is pressed, we wish to offer a group of 4 more items. In order to get to this group when F1 is pressed, the "Next solution" for the original help screen must be defined as this new group. Groups are notated by a leading "\$" in the solution string. Every help file must have one and only one "Super Group" which is notated by two leading dollar signs ("\$\$"). Help will automatically go to the SuperGroup if a "Next solution" is not provided.

In the help editor open the Video help file and get into the "Text" editing mode. Find the solution string for the menu item in question (f518). Once this record has been found, we must now enter a string in the "Next solution" editfield. We will arbitrarily call this string "world(n)X1", making sure that it is preceded by "\$", denoting the solution as a group.

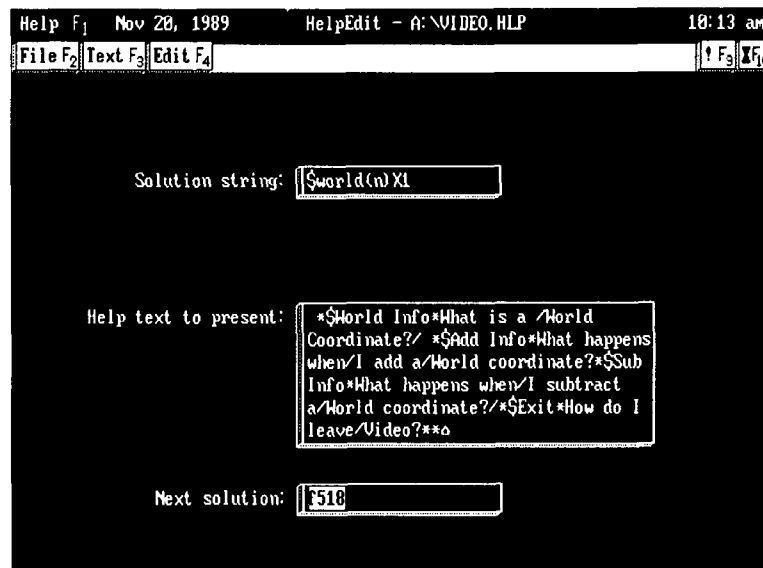


Once the solution string is entered and saved, choose the F2 "Switch to groups" option to define the group and enter the solutions for each of the items in the group.

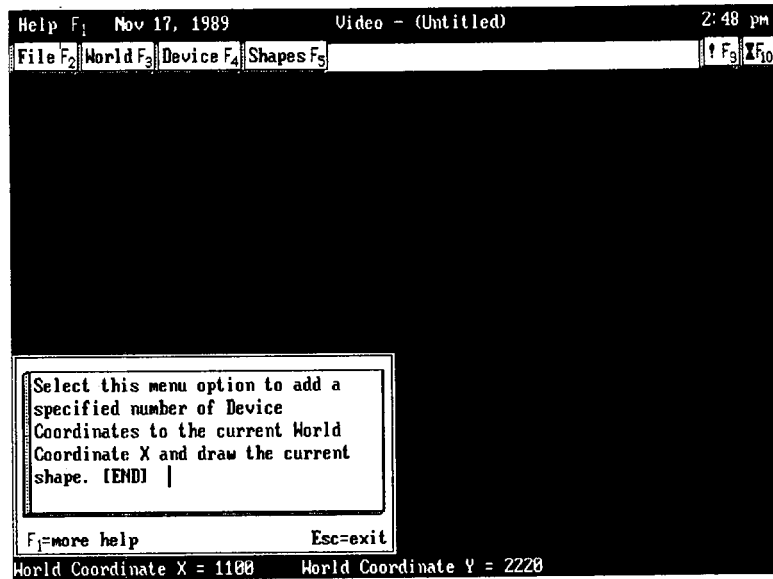
The solutions can be more groups or other text solutions. The group editor also gives you the option of numbering the items in a group for procedural instructions.



To inspect the group strings, you may also "Switch to text" and then select "Show groups" to see how the group is actually defined and how each field in the group is arranged with its delimiter and text solution. The "/" delimiter is used to start a new line in the topic.



The "Next solution" string for a group can reference another group, a specific help text, or any other solution. In this example, the "Next solution" refers to the original menu option help. Now when F1 is pressed on the "Next (n) X" option, the original help text will appear with a "F1=more help" prompt. When F1 is pressed now, the defined group will appear on the screen.



Help Rule Base Utility - DMHELP.UTL

When you use this utility, you should rename it `DMHELP.ACC` and run it as the help accessory from your application. You can then press F1 in different context within the application and examine what is in the information queue. The utility will open the application help file, if there is no help file one will be created in the current directory.

Adding menu bar text

To enter in help topics, use the Add menubar text option from the File F2 Menu. This option will add entries for the application menu options only. It will enter the solution string and the menu option return code for each menu option. You then update these records by assigning a topic string and the help text for each option using the help editor, `DMEDITOR.PDM`.

Adding new rules

The premises (IF entries) are displayed on the main screen. A rule can have up to 8 premise lines each ANDed together. The conclusion (THEN entry) is the only field you enter. Add a premise by selecting one of the queue items in the list box and pressing ENTER. The F4 menu options become available when a queue item is selected. Remember that you must first get the information registered in the queue by getting your application into the state you want to give help on.

Group numbers

Group numbers are used to categorize rules. Depending on the state of the help queue, rules with specific group numbers will be searched first. The following group numbers and their functions are available.

Group Number	Function
0	This rule group will always be scanned if a conclusion was not found in groups 3 through 11.
1	This rule group is scanned if a group 0 rule's premise contains a solution.
2	This rule group is scanned last.
3	This rule group contains dialog box rules.
4	This rule group contains edit field, push button, check box, radio button, icon button, and application-defined component rules.
5	This rule group contains message box rules.
6	This rule group contains menu rules.
7	This rule group contains menu option rules.
8	This rule group contains command event or accelerator rules.
9	This rule group contains list box rules.
10	This rule group contains "what application is running" rules.

11 This rule group contains state and queue rules added by the application, see the **ihm_new_entry** and **ihm_add_state** functions in the Help Manager section of the DeskMate Technical Reference.

Internal This rule group contains the IHM internal rules.

One of the groups 3 through 11 is always searched first, followed by group 0 and possibly group 1. If a conclusion is not found, the internal rules are searched and then finally group 2. If none of the rules result in a conclusion, the help manager will "fire" the \$\$<SuperGroup> text solution and provide general help.

This utility, **DMHELP.UTL**, creates only group 0 rules. Use the help editor, **DMEDITOR.PDM**, to change the group numbers.

Queue Data

You can use any queue entry displayed in the list box as a premise to a rule. Each premise built from a queue entry has the associated queue entry number stored with it. This means a rule built from the queue is time-related, it depends on the order in which events occur.

Functions

The Functions F4 Menu allows you to make numeric comparisons to the data in a numeric edit field, to negate the results of a premise, or to call a function.

TEST Use this function to make numeric comparisons on data in a numeric edit field. Format characters for an edit field or the number of decimal points are NOT stored in the queue.

TEST puts 4 in the keyword field

NOT TEST puts 5 in the keyword field

Put the corresponding function number in the Variable # field

Put the call's parameters into the Value string field. If there is more than one parameter, single space between them.

This function is available in two places, the first is when the queue entry is a dialog box and the second is when the queue entry is a single edit field which is currently running. In a dialog box, the edit field's tab number in the box is required to identify the edit field. For the single edit field, the tab number is zero. The parameters should be placed in the value field of the premise.

The TEST functions available are:

1 EditFieldGT(Tab#, "<number string>")

Test if greater than.

2 EditFieldLT(Tab#, "<number string>")

Test if less than.

3 EditFieldEQ(Tab#, "<number string>")

Test if equal to.

NOT Use the NOT clause to negate the results of the entire premise line. For example, "NOT running.dlg.box Copy File" means: running any dialog box except the Copy File box.

CALL Some queue entries have data associated with them. Function calls are available that allow a rule to access that data. These calls are designed to make data and state comparisons.

CALL puts 2 in the keyword field

NOT CALL puts 3 in the keyword field

Put the corresponding function number 1 through 11, defined below, in the Variable # field .

Put the call's parameters into the Value string field. If there is more than one parameter, single space between them.

The instructions below also indicate how to enter a call premise if you use the help editor DMEDITOR.PDM to enter the rule yourself. The function calls available are:

1 EditFieldEmpty(tab #)

This function can be chosen in two places. The first is when the queue entry is a dialog box, the edit field's tab number is required. The second is when the queue entry is an edit field, leave the value string field EMPTY. The function will check to see if the string is null.

2 EditFieldCmp(tab #, tab#)

This function can be chosen when the queue entry is a dialog box with more than one edit field. The function will compare the edit field strings to see if they are the same.

3 ListBoxEmpty(tab #)

This function can be chosen in two places. The first is when the queue entry is a dialog box, the list box's tab number is required. The second case is when the queue entry is a single list box, leave the value string field EMPTY. The function will check to see if the list box is empty.

4 ScanMessage("<string>")

This function can be chosen when the queue entry is a message box. The function will scan up to a 20 character message saved for the <string>.

5 CheckBoxChecked(Tab #)

This function can be chosen in two places. The first is when the queue entry is a dialog box, the check box's tab number is required. The second is when the queue entry is a check box, leave the value string field EMPTY. The call will check to see if the check box is selected.

6 IconButtonSelected(Tab #)

This function can be chosen in two places. The first is when the queue entry is a dialog box, the icon button's tab number is required. The second is when the queue entry is an icon button, leave the value string field EMPTY. The call will check to see if the icon button is selected.

7 RadioButtonSelected(Tab#, Ordinal#)

This function is available when the queue entry is a dialog box only. The call will check to see if the specified radio button (Ordinal #) within the specified group (Tab #) is selected.

8 MenuItemGreyed()

This function is available when a menu is pulled down, the user selects an item, and presses F1. The last entry made in the queue will indicate if the item is enabled or disabled.

9 ClipboardEmpty()

This function is available at all times. The function checks the clipboard contents. The function is also available through the Functions Menu, Check clipboard option.

10 PreviousKeyStrokes()

This function is available at all times. The function will check if any keystrokes or mouse events occurred before the current queue entry. The function is also available through the Functions Menu, Previous keystrokes option.

11 KeyStrokesDuring()

This function is available at all times. The function will check if any keystrokes or mouse events occurred during the current queue entry. The function is also available through the Functions Menu, Keystrokes during option.

DeskMate Help Editor - DMEDITOR.PDM

The help editor allows you to enter the information the user will see when help is requested. Specific help given in the help window, help topics displayed on topic buttons, and procedural help displayed on numbered buttons is all entered and edited using this tool. This documentation assumes you will edit the help file created by DMHELP .UTL.

Working with rules

The default screen shows the current set of rules in the file. One rule is displayed at a time, use the First, Next, Previous, and Last menu options from the F3 Menu to view other rules.

Editing a rule

The display shows the internal fields of a rule, the group number, the conclusion, and each premise. Use the Insert menu option in the F3 Menu to add a new premise to the rule being viewed.

Adding a new rule

You can also add a new rule by selecting New from the F3 menu. If the rule's conclusion is a help topic, the rule and its premises must be in group 0. If not, the rule WILL NOT fire.

Rules that chain

If the group number is 1 then this is a chaining rule. That is, this rule will fire when a group 0 premise matches this rule's conclusion. This means that in order to prove the premise of the group 0 rule, the group 1 rule must be proven as well.

Working with displayed text

To modify the text displayed use the Switch to text menu option in the File F2 Menu.

Size restrictions

The solution string can be up to 20 characters long.

The help text for one topic can be up to 900 bytes long.

Link to rules

The help text solution string must match a conclusion in one of the rules or a conclusion returned by the internal rules. An entry is automatically added for any rule created with the DMHELP .UTL utility. Rules added with this editor must also have a corresponding text entry added.

Working with solution extensions

A menu options's solutions is its return code, a hexadecimal number, for instance F158. For other entries in the queue, attach the following extension

- Menu Name.mm (for main menu)
- Dialog Box Title.db
- List Box Title.lb
- Message Box Title.mb
- Menu Option.g (for a grayed or disabled menu option)

Working with button topics

Use the Switch to groups menu option in the File F2 Menu to display the button topics and their corresponding text. Use the Edit Topic option to give the topic a three line title, a solution string, and select whether or not to number the button.

Help File Compression Utility - TOKEN.PDM

This utility replaces common strings in the application help file with a number or *token*. The help file is made smaller on the disk and is untokenized as it is accessed. To tokenize a file, do the following:

- 1) Create your application help file, MYAPP.HLP, using the Help tools, DMHELP.UTL and DMEDITOR.PDM.
- 2) Build an ASCII token file, MYAPP.TKN, that contains the list of tokens for your help file. Use the following guidelines for building your token file.

- a) The token records must fit within two database pages which are 1K bytes/page in size. The format of each token record is

Token(N), Field Delimiter, Record Delimiter

where N in Token is the number of bytes in the Token, and the Delimiters are each one byte.

- b) Token strings should be 3 bytes or longer.
- c) Each token should be separated by a CR/LF, EXCEPT for the last token which should terminate the file WITHOUT a CR/LF.
- d) You must manually create a file of token strings for the TOKEN.PDM compression utility. The following is a procedure to accomplish this task:

1. Use the help editor to retrieve the application help file information. Use the F2 Menu to "Switch to text" and then "Print text..." to a file MYAPP.TKN.
2. Open the file with your text editor. At the beginning of each page in MYAPP.TKN is a header describing the file, for instance,

Help Database Text in C:\APPHelp\MYAPP.HLP

Delete all occurrences of this header. On the left of the page is a list of all the solutions. Delete these solution strings and the "Solutions" header. Also delete the "Text" header at the beginning of each page. Your file should now only contain the actual help text.

3. On the occurrence of every string three or more characters in length, do a "search and replace" of the string, replacing it with a space, or nothing. If your editor displays the number of replacements made, note that number. If there were at least three occurrences of the string, then you may enter it as a string to tokenize. Each string must appear alone on a separate line, terminated by a carriage return/line feed. As you repeat this process for every candidate string, the blanks you have created will shrink the size of the file.

4. In the case of strings being substrings of longer strings (such as in plural versions of a word), do a search for the longer string first, noting the number of occurrences. If the longer string only occurs twice and the substring occurs four times, use the substring as the string to tokenize. For example,

You search for the string "removed" and replaced it with nothing, noting 2 occurrences. This is not enough to take advantage of compression. However, doing a search and replace of the string "remove" produced 2 occurrences. So you know that the string "remove" occurred 4 times in the file, allowing TOKEN.PDM to do a slight compression.

Since you are replacing strings with nothing as you go along, it is important to replace the longer strings first if possible, in order not to miss the opportunity to use both the longer string and the substring as tokens.

TOKEN.PDM distinguishes case, so capitalized versions of words will need to be included as well as the lower-case word.

- 3) Run the DeskMate application TOKEN.PDM giving it the name of your application help file, MYAPP.HLP. The help file and token file, MYAPP.TKN, must be in the current directory with the utility.
- 4) Token creates a MYAPP.LST file when it has tokenized the file. This file lists the number of bytes each token saved. If the number of bytes for a token is less than or equal to zero, remove the token and retokenize the file. Remember to always retokenize from the original untokenized file.

Help File Format

In order for you to better understand what information is stored in your help database file, the database format for text and rules is outlined below.

There are two tables in your database file. The first is a TEXT table. It has the following columns:

Sol	This is the Solution TAG. It is the solution that a rule produced.
Text	The actual help that goes with this topic.
F1	The next help window to chain to when the user presses F1.

The second table is the table RULES. It has the following columns:

Grp	A rule group. Group 0 is always the group in which rules are placed that will give a solution. If these rules cause any other rules to fire, these rules should be in another group.
Rule#	The number of the rule. The rules are sorted by rule number, because you may want to try to fire one rule before another.
PorC#	A premise or conclusion number. The records in the table are also sorted by PorC#. The conclusion is number 0 because it is the first thing pulled out of the table when the inference engine starts to fire a rule. (This is because it is doing Backward Chaining- start with the conclusion , and then prove the premises).
Var#	An identifier for the engine to do faster searching. Each new variable added to the table has a unique identifier. The variables are DeskMate components, like a dialog box.
Var	The variable field is a string representing a variable for which a value is expected such as "running.cmp" (DISABLED FIELD)
Value	The String field for which contains a value. The if the value is the current one for the variable, then the premise line will succeed.
Bind	A variable to bind to like "X". If a rule has a binding, the variable will bind to the currently known value of the variable. This eliminates repetition of rules that do the same exact thing. Only premises can bind.
KeyW	A number indicating the negation (NOT) of a premise, or a TEST or CALL. TEST will do number comparisons, and assumes the value string field is a numerical value. CALL is used to execute a pre-defined function. The parameters of the function are placed in the value field.

Q#

The number in the queue for which a premise line test applies. If the queue number = 0, it is assumed the premise does not use predefined facts.

Part 6
Writing Tutorials and Demos

"Writing Tutorials and Demos"

Contents

The DeskMate Tutorial Technology	6-1
Authoring a Tutorial Script	6-3
The DeskMate Introductory Tutorial - DMINTRO.TUT	6-5
DMINTRO.DOC	6-6
INTRO2.DOC	6-14
INTRO3.DOC	6-19
OPTIONS.DOC	6-25
MOUSE.DOC	6-27
MOUSE1.DOC	6-37
Script Command Reference	6-41
ALLOW_INHIBIT	6-43
CALL	6-44
CHANGE_DIR	6-45
COUNT_ABOVE	6-46
COUNT_BELOW	6-47
COUNT_DEC	6-48
COUNT_EQUAL	6-49
COUNT_INC	6-50
COUNT_SET	6-51
DELETE_DIR	6-52
DELETE_FILE	6-53
DISK_SPACE	6-54
ESC_FLAG	6-55
EXPECT_KEY	6-56
FILE_EXIST	6-57
GET_ARROWS	6-58
GET_DLGBOX_CMP	6-59
GET_KEY	6-60
GET_LB_ITEM	6-61
GET_RB	6-62
GET_TEXT	6-63
GET_TO_MENU	6-64
GOTO	6-65

IF_FALSE_GOTO	6-66
IF_TRUE_GOTO	6-67
IGNORE_INHIBIT	6-68
IN_DLGBOX	6-69
IN_FILE	6-70
IN_LISTBOX	6-71
IN_MSGBOX	6-72
INVERT_OFF	6-73
INVERT_ON	6-74
KEY_INTERVAL	6-75
KEY_WITHIN	6-76
LOOP_TO	6-77
M_PROMPT_ORG	6-78
MESSAGE_BUFFER	6-79
MESSAGE_OFF	6-80
MESSAGE_ON	6-81
ON_QUIT_GOTO	6-82
ON_TIMEOUT_CALL	6-83
OPTIONS	6-84
PASS_KEY	6-85
PASS_WHILE	6-86
PAUSE_MODE	6-88
PICTURE_OFF	6-89
PICTURE_ON	6-90
POINT_TO	6-91
PRESERVE_DT_CFG	6-92
PRESERVE_FILE	6-93
PROMPT	6-94
RESTORE_DT_CFG	6-95
RESTORE_FILE	6-96
RETURN	6-97
RUN_RESOURCE	6-98
START_IN	6-99
TAG	6-100
UNPACK_FILE	6-101
Keystroke Definitions	6-103

Tutorial Player - PLAY.PDM/DMPLAY.RES	6-105
Demo Launcher - DEMO.PDM	6-106
Event Recorder - RECORD.PDM/DMRECORD.RES	6-107
General Rules of Recording	6-107
Script File Interpreter and Compiler - DMEI.EXE/DMEC.EXE.....	6-109
Tutorial Compression Tools - DMPACK.EXE/DMUNPACK.RES	6-110

The DeskMate Tutorial Technology

The DeskMate Tutorial Technology allows a programmer to create store-demos and application tutorials which use the actual application to execute. A tutorial interacts with the user, allowing the user to enter information and perform tasks which teach the user how to operate the application. A demo demonstrates the functions of the application, explaining key functions of an application without the user's intervention. The same technology is used to build and execute both demos and tutorials. Tutorials will be discussed in this documentation since demos are simply a tutorial without user interaction.

Tutorials are written using *scripts* which are compiled into *event* files. Pictures and data files needed by the tutorial are stored along with the event files in a compressed *tutorial* file. The tutorials are launched from the Play portion of the technology. The application `PLAY.PDM` and its resource, `DMPLAY.RES`, read in the tutorial file, extract files as needed or directed, interpret the commands in the scripts, and send events to the application.

A comprehensive example of a tutorial is also included in this section. The DeskMate Introductory Tutorial, `DMINTRO.TUT`, covers the key elements of a tutorial and makes a good template or guide when writing your own tutorial. The actual "source" for the tutorial resides in the `SAMPLES\DMINTRO` directory.

The script language used to write scripts is defined in the Script Command Reference. The documentation for each of the tutorial tools provided with the kit follows the command reference.

Authoring a Tutorial Script

Step 1 - Create the Storyboard

The first step in developing a tutorial is deciding what the tutorial should teach. Tutorials should be broken down into *lessons* the user can take at different times. The first lesson should be introductory in nature, the following lessons should cover specific areas of your application, each lesson increasing in difficulty and building on the information learned in the previous lessons. The number of lessons you supply with your application is dependent on how much time you want to spend on the tutorial and how much disk space you choose to dedicate to this on-line documentation.

Step 2 - Record the Storyboard

Once you have decided what your tutorial will cover and how it will be organized you can *record* the initial events by using the Record portion of the technology. This step is optional and once you are familiar with the structure of events and the authoring of scripts you probably will not want to record the storyboard to create a tutorial. Once the first lesson is written, you will probably use it to build the next one and so on. Use the application RECORD .PDM and its resource, DMRECORD .RES to record the events you want executed in your tutorial. Use the DMEI .EXE utility to create the initial script file. You will want to strip out most of the information recorded and concentrate on the actual events recorded.

Step 3 - Develop and Test the Tutorial

Once the initial tutorial script file is built, you will then go into the development process which involves editing the script, compiling it into a event file, packing the files into a tutorial file and actually running the tutorial. This is the longest process in building a tutorial. You may want to make changes to the storyboard after using the tutorial. Considerable error checking and processing must be done to insure the user does not perform actions which your tutorial does not handle correctly.

Refer to the DeskMate Introductory Tutorial for examples of error handling and other tutorial functions. Use the Script Command Reference for information about all the function commands available in the tutorial technology.

Script Rules and Guidelines

The script files must contain only printable ASCII characters and 0DH for carriage returns.

The TAB character is not allowed.

Commands and their parameters may be delimited by a space, comma, or carriage return. The command descriptions use a space delimiters when the parameters are listed on the same line as the command and a carriage return for string parameters. Multiple commands appearing on the same line are delimited with commas.

Comment lines follow the C language convention, /* */. Embedded comment lines are not supported.

Text information, such as data for edit fields in dialog boxes is enclosed within brackets, { } and should be on a line or lines by itself.

Never alter a user's data file.

For example, the Address Book tutorial does not alter a user's address book data file, the `COPY_FILE` command is used to save the original file before the tutorial begins.

Always make sure there is enough disk space on the diskette to run the tutorial before it begins.

The `DISK_SPACE` command is used to insure that the player has enough room on the disk to create a working file.

Pauses should be added before the `RETURN_KEY` keystroke when selecting menu options so that the viewer can detect which menu option was chosen. A delay of half a second, 50 counts, is usually sufficient.

Tutorials should always tie up any loose ends prior to terminating, this includes the removal of working files used by the tutorial.

The `CHANGE_DIR` and `DELETE_DIR` commands are used to cleanup the directory structure when a tutorial ends in an unknown state. `QUIT_OK` should be placed into the event sequence at any point where it is permissible for the tutorial to end. Remember `ESC` will proceed until the next `QUIT_OK` is encountered.

Tutorial Guidelines

The maximum size of an individual event file is 10K bytes. Scripts which produce larger event files should be divided into separate scripts and the event files should be "chained" together.

The maximum size of a picture in a tutorial is 8K bytes. Larger pictures will not be displayed and an error condition will not be returned.

Printing and task switching are not supported in tutorials.

The DeskMate Introductory Tutorial - DMINTRO.TUT

The DeskMate tutorial DMINTRO.TUT was built using the Tutorial Technology. The files needed to create the tutorial are included in the SAMPLES\DMINTRO directory. The tools needed to build the tutorial are in the TOOLS directory.

The MK.BAT is used to build the tutorial file.

```
del dmintro.tut
dmec dmintro
dmec intro2
dmec intro3
dmec options
dmec mouse
dmec mouse1
dmpack dmintro.tut dmintro.evn intro2.evn intro3.evn options.evn bob.dft
dmpack dmintro.tut desktop.dft desktopd.dft dmcorkbd.dft desktext.fig
dmpack dmintro.tut mouse.evn mouse1.evn listbox.fig arrow.fig mouse.fig rat.fig
dmpack dmintro.tut help2.fig compass.fig powerful.fig superfl.fig magnify.fig
dmpack dmintro.tut start.fig waiter.fig explorer.fig
dmpack dmintro.tut /u findtut.fig
```

The tutorial uses the following event files (.EVN) , picture files (.FIG), and data files (.DFT):

DMINTRO.EVN , INTRO2.EVN, INTRO3.EVN, OPTIONS.EVN, MOUSE.EVN,
MOUSE1.EVN

LISTBOX.FIG, MOUSE.FIG, ARROW.FIG, DESKTEXT.FIG, RAT.FIG, FINDTUT.FIG,
COMPASS.FIG , POWERFUL.FIG, MAGNIFY.FIG, SUPERF1.FIG, HELP2.FIG,
EXPLORER.FIG , WAITER.FIG, START.FIG

BOB.DFT, DESKTOPD.DFT, USERDICT.DFT, DESKTOP.DFT, DMCORKBD.DFT

DMINTRO.DOC

```
PRESERVE_DT_CFG
/*-----*/
/* Introductory Lesson : Introduction, Part Menu, and Part 1 */
/*-----*/

PICTURE_ON 0 0 0 "FINDTUT.FIG"

PRESERVE_FILE 0 "desktopd.cfg"

TAG "stut"
COUNT SET 9 0
CTRL_U

ON QUIT GOTO "end lesson"
SET HELP 1
GOTO "options" IN_FILE "options.evn"

/*-----*/
/* clean up form parts */
/*-----*/

TAG, "menu"
MESSAGE_OFF PICTURE_OFF

COUNT EQUAL 9 0
IF_TRUE_GOTO "menu1"

RESTORE_FILE 0 "dmcorkbd"

TAG "menu1"

/*-----*/
GOTO "show_menu" IN_FILE "options.evn"
/*-----*/

/*-----*/
/*                               P A R T   O N E                               */
/*-----*/

TAG "Section 1"
ON_QUIT_GOTO "menu"

M PROMPT ORG 2200 4400
PICTURE_ON 1 1900 1760 "start.fig"
EXPECT_KEY {c}

M PROMPT ORG 2200 2750
POINT TO 450, 250, 1, 2 /* point up */
PROMPT 1 1400, 2310, 5200, 1100 {c} "C=continue"
"The top line on your screen is the \C1title line\C0. The
title line contains four items of information. The
first item reminds you which key to press when you
need Help. You will learn all about Help in Part 2
of this lesson."
MESSAGE_OFF

POINT TO 1600, 250, 1, 2 /* point up */
PROMPT 1 1200, 2640, 5500, 220 {c} "C=continue"
"The second item on the title line is the date."
MESSAGE_OFF
```



```

POINT TO 4100, 250, 1, 2 /* point up */
PROMPT 1 1200, 2530, 5500, 660 {c} "C=continue"
"The third item of information is the application name
and \C\current path\C0. The current path is where DeskMate
is currently looking for data files on your disk."
MESSAGE_OFF

POINT TO 7400, 250, 1, 2 /* point up */
PROMPT 1 1200, 2640, 5500, 220 {c} "C=continue"
"The last item on the title line is the time."
MESSAGE_OFF

POINT TO 2400, 1760, 1, 2 /* point up */
PROMPT 1 1100, 2640, 5800, 660 {c} "C=continue"
"The Desktop displays \C\application boxes\C0. Each application
box contains the name of one DeskMate application. This
box represents the Address Book application."

PROMPT 1 1200, 2520, 5500, 880 {c} "C=continue"
"The Address Book application is used to store and
organize information about people and businesses. You
can use the Address Book to print mailing labels and
form letters."
MESSAGE_OFF

POINT TO 4000, 1760, 1, 2 /* point up */
PROMPT 1 1200, 2640, 5600, 440 {c} "C=continue"
"Next is the Calendar application box. The Calendar
application helps you organize your personal schedule."
MESSAGE_OFF

POINT TO 5600, 1760, 1, 2 /* point up */
PROMPT 1 1200, 2640, 5500, 440 {c} "C=continue"
"PC-Link is a telecommunication application that
provides access to a vast database of information."
MESSAGE_OFF

POINT TO 800, 2090, 1, 2 /* point up */
PROMPT 1 1200, 2530, 5500, 660 {c} "C=continue"
"Another type of application box appears in the form
of a list box. Using the arrow keys, you can scroll
through and choose from the items listed."
MESSAGE_OFF

POINT TO 1400, 770, 1, 0 /* point left */
PROMPT 1 1200, 2520, 5500, 660 {c} "C=continue"
"The top of an application list box contains the name
of the application. The Text application is used to
create and edit documents."
MESSAGE_OFF

POINT TO 1500, 1430, 1, 0 /* point up */
PROMPT 1 1200, 2640, 5500, 440 {c} "C=continue"
"The bottom contains the names of the files stored on
your disk that were created by the application."
MESSAGE_OFF

POINT TO 1100, 935, 1, 2 /* point up */
PROMPT 1 1200, 2520, 5600, 660 TAB KEY "move highlight"
"A \C\highlight\C0 is used to select The DeskMate application
that you wish to run. Currently, the Text application is
highlighted. Press TAB to move the highlight."
MESSAGE_OFF
PASS_KEY

```



```

POINT TO 2400, 1540, 1, 2 /* point up */
PROMPT 1 1100, 2420, 5700, 880 {c} "C=continue"
"Great! The Address Book application is now highlighted.
Pressing ENTER will run the currently highlighted
application. You will run an application in Part 3 of
this lesson."
MESSAGE_OFF

POINT TO 7200, 2100, 1, 2 /* point up */
PROMPT 1 1200, 2520, 5500, 660 {c} "C=continue"
"The Programs list box is unique; it displays a list
of all the applications on your disk. You can run any
application using the Programs list box."

PROMPT 1 1200, 2310, 5500, 1100 {c} "C=continue"
"The Programs list box is very useful because not
every application is represented on the Desktop with
an application box. In the lesson about Desktop, you
will learn how to customize your Desktop by adding
and deleting application boxes."
MESSAGE_OFF

POINT TO 1600, 3960, 1, 3 /* point up */
PROMPT 1 1900, 2310, 5200, 1100 {c} "C=continue"
"This is the Teach Me application box. When you run
Teach Me, it presents you with a list of lessons.
Each lesson will guide you through one of the
DeskMate applications. You will learn how to run
Teach Me in Part 4 of this lesson."
MESSAGE_OFF

POINT TO 6400, 3960, 1, 3 /* point up */
PROMPT 1 1300, 2310, 4500, 1100 F10 "press F10"
"The final box is the Month \Claccessory\C0.
Accessories are special tools that can be
run anywhere within DeskMate. The F10 \Clmenu\C0
contains all the accessories that are
included with DeskMate."
MESSAGE_OFF
PICTURE_OFF
PASS_KEY

M_PROMPT_ORG 2200 4400
PICTURE_ON 1 700 1760 "waiter.fig"
EXPECT_KEY {c}
PICTURE_OFF
M_PROMPT_ORG 2200 2750

PROMPT 1 1700, 2310, 4100, 660 {c} "C=continue"
"This is a menu. To pull down a menu, you
press the function key appearing after
the menu's name."

PROMPT 1 1700, 2310, 4100, 880 LEFT_ARROW "use \5\6"
"Once a menu is pulled down, you can use
\5\6 and \6\7 to view neighboring menus.
Use \5\6 to view all the Desktop menus.
Stop when you get to the File menu."

```



```

COUNT SET 5 6
TAG"MENU KEYS"
PASS KEY
PROMPT 1 2400 3080 3200 220 ABORT_KEY "use \5\6"
"You're doing fine! Keep going."
EXPECT KEY LEFT_ARROW
LOOP TO 5 "MENU_KEYS"
PASS_KEY

MESSAGE OFF
POINT TO 1600, 715, 1, 0 /* point up */
PROMPT 1 1200, 2750, 5600, 660 {c} "C=continue"
"Some options in a menu might be \Clshadowed\C0. The GET INFO
option in the File menu is shadowed. A shadowed
option is one that is not currently available."
MESSAGE_OFF

POINT TO 2650, 2035, 1, 0 /* point up */
PROMPT 1 2900, 2750, 4400, 1100 DOWN_ARROW "highlight option"
"Choose a Menu Option\r\n
To choose a menu option you highlight it
and then press ENTER. \A\B and \8\9 are used
to move the highlight. Use \A\B to highlight
the UPDATE SCREEN option."

COUNT SET 5 5
TAG"US KEYS"
PASS KEY
PROMPT 1 2400 3300 3200 220 ABORT_KEY "highlight option"
"You're doing fine! Keep going."
EXPECT KEY DOWN_ARROW
LOOP TO 5 "US_KEYS"
PASS_KEY

MESSAGE OFF
PROMPT 1 2900, 2640, 4500, 1320 RETURN_KEY "press ENTER"
"The UPDATE SCREEN option is used to update
the contents of all the application list
boxes whenever you insert a different disk.
Press ENTER to choose the menu option.
Because we are merely teaching you about
menus, you SHOULD NOT insert another disk."
MESSAGE OFF
PICTURE_OFF
PASS KEY
/* ON_QUIT_GOTO "Q" */

POINT TO 400, 660, 1, 2 /* point up */
PROMPT 1 3000, 2420, 4500, 880 F2 "pull down menu"
"Did you notice that the screen was redrawn?
Let's look at the File Menu again. Use the
function key following the menu's name to
pull down the menu."
MESSAGE OFF
PICTURE_OFF
PASS_KEY

POINT TO 2650, 2035, 1, 0 /* point up */
PROMPT 1 2900, 2750, 4300, 1100 {c} "C=continue"
"Notice that \"CTRL+U\" appears next to
the UPDATE SCREEN option. This is an
\Claccelerator key\C0. Accelerator keys allow
you to choose a menu option without
pulling down the menu."

```



```

MESSAGE_OFF

ESC FLAG 1
PROMPT 1 3000, 2640, 4500, 660 ESC KEY "press ESC"
"Let's use the accelerator to update the
screen again. First, remove the File menu
from the screen by pressing ESC."
PASS KEY
ESC FLAG 0
PICTURE OFF
/* ON_QUIT_GOTO "Q" */

PROMPT 1 1200, 2640, 5500, 440 CTRL U "use accelerator"
"Press CTRL+U to update the screen without pulling
down the File menu."
PICTURE OFF
MESSAGE_OFF
PASS_KEY

PROMPT 1 1100, 2640, 5800, 440 {c} "C=continue"
"Sometimes menu options need more information to complete
a task. When this occurs a dialog box will appear."
MESSAGE_OFF

M PROMPT ORG 2200 3960
PICTURE ON 1 2100 1870 "explorer.fig"
EXPECT KEY {c}
PICTURE_OFF

PROMPT 1 1600, 3080, 4800, 220 ABORT KEY "choose menu option"
"Choose the FORMAT option from the Disk menu."
GET_TO_MENU 4, 2, "Disk" "FORMAT" SPACE_KEY

POINT TO 3450, 1650, 1, 1 /* point right */
PROMPT 1 1200, 3850, 5500, 440 {c} "C=continue"
"The format option is used to format new diskettes so
that you can start storing information on them."
MESSAGE_OFF

POINT TO 3650, 2090, 1, 1 /* point up */
PROMPT 1 1100, 3850, 5800, 440 {c} "C=continue"
"The box next to the Drive: prompt is where you will type
the name of the drive you wish to use to format a disk."
MESSAGE_OFF
PICTURE_OFF

PROMPT 1 1200, 3850, 5500, 660 {a} "type \"A\""
"The flashing cursor indicates that the box is active
and waiting for you to type a response. Let's pretend
that we are going to format a disk on drive A."
PASS_KEY

PROMPT 1 1200, 3850, 5500, 440 BKSPACE KEY "press BACKSPACE"
"The letter \"A\" appeared in the box. The BACKSPACE
key is used to correct mistakes. Try it."
PASS_KEY

PROMPT 1 1200, 3850, 5500, 440 TAB KEY "move cursor"
"The letter \"A\" was removed. The TAB key is used to
move the cursor to the next component. Try it."
MESSAGE_OFF
PASS_KEY

```



```

POINT TO 5100, 2530, 1, 0 /* point up */
PROMPT 1 1200, 3850, 5500, 880 TAB KEY "move cursor"
"The cursor now appears in the box next to the Options
prompt. You do not need to type any options to do a
standard format. Use the TAB key to move the cursor
to the next component."
MESSAGE OFF
PASS_KEY

POINT TO 2700, 2970, 1, 1 /* point up */
PROMPT 1 1200, 3850, 5600, 660 SPACE KEY "press space bar"
"This is a \C1check box\C0. If you want the diskette to
be a bootable diskette, you would check the INSTALL
OPERATING SYSTEM box by pressing the space bar."
MESSAGE OFF
PASS_KEY

PROMPT 1 1100, 3850, 5700, 660 SPACE KEY "press space bar"
"The box is now checked. Since disks used to store files
created by DeskMate do not need the operating system,
uncheck the box by pressing the space bar again."
MESSAGE OFF
PASS_KEY

POINT TO 3100, 3355, 1, 1 /* point up */
PROMPT 1 1200, 3850, 5500, 880 TAB KEY "move cursor"
"When you are done providing all the information to
complete the task of formatting a disk, you would
push the OK button. Move the cursor to the OK button.
Can you guess what key to use?"
MESSAGE OFF
PASS_KEY

POINT TO 5000, 3355, 1, 0 /* point up */
PROMPT 1 1200, 3850, 5500, 440 TAB KEY "move cursor"
"Since we are not actually going to format a disk,
push the CANCEL button instead."
MESSAGE OFF
PICTURE OFF
PASS_KEY

PROMPT 1 1200, 3850, 5500, 220 SPACE KEY "push CANCEL"
"Press space bar to push CANCEL and abort the format."
PASS_KEY

M PROMPT ORG 2200 2750
PROMPT I 1200, 2310, 5500, 1100 {c} "C=continue"
"The ESC key could also be used to cancel a dialog
box. The ESC key is your exit key; it will quit
applications, menus, and dialog boxes. If you press
ESC now, you would exit DeskMate and return to DOS.
Now, let's review what you have learned."

```



```
PROMPT 1 1625, 1540, 4700, 2420 {c} "C=continue"
"
\C1Part 1: Lesson Summary\C0\r\n
- The Desktop is used to run DeskMate\r\n
  applications.\r\n
- Accessories are special tools that can run\r\n
  anywhere within DeskMate.\r\n
- The F10 menu contains all the accessories.\r\n
- Title line provides information about\r\n
  DeskMate and the date and time.\r\n
- Menus contain the options available in an\r\n
  application.\r\n
- Menus are displayed using function keys."
```

```
PROMPT 1 1625, 1430, 4700, 2640 {c} "C=continue"
"
\C1Part 1: Lesson Summary\C0\r\n
- To choose a menu option, you pull down\r\n
  the menu, highlight the option, and then\r\n
  press ENTER.\r\n
- Accelerator keys are a quick way to choose\r\n
  a menu option.\r\n
- Dialog boxes ask you for information to\r\n
  complete a task.\r\n
- TAB moves the cursor to the next choice.\r\n
- Space bar checks boxes and pushes buttons.\r\n
- ESC exits menus, dialog boxes, and\r\n
  applications."
```

```
MESSAGE OFF
GOTO "mēnu"
```

```
/*.....*/
```

```
TAG "get_help_arrows"
ESC_FLAG 1
```

```
GET KEY
KEY WITHIN DOWN ARROW []
IF FALSE GOTO "gha_not_dn_arrow"
PASS KEY
COUNT DEC 0
COUNT ABOVE 0 0
IF TRUE GOTO "get_help_arrows"
```

```
TAG "gha_no_more"
```

```
M PROMPT ORG 2100 2420
PROMPT 1 2000 2200 5000 440 ABORT KEY "press ESC"
"There are no more lines of information. Press
ESC to exit Help."
PAUSE MODE 200
EXPECT KEY ESC KEY
MESSAGE OFF PICTURE_OFF
ESC_FLAG 0
```

```
TAG "gha_exit"
```

```
PASS KEY
ESC_FLAG 0
```

```
RETURN
```

```
TAG "gha_not_dn_arrow"
```

```
MESSAGE OFF
PICTURE OFF
KEY_WITHIN ESC_KEY []
```



```

IF TRUE GOTO "gha_exit"
COUNT BELOW 0 3
IF TRUE GOTO "gha_no_more"

M PROMPT ORG 2100 2420
PROMPT 1,2000,2200,5000,440,ABORT KEY,"read information"
"Press ESC to leave Help or press \A\B to continue
reading the information."

GOTO "get_help_arrows"
/*.....*/
TAG "e2"
ESC KEY
TAG "e1"
ESC KEY
GOTO "menu"

TAG,"no_space"

PROMPT 1 2000 1760 4000 660 {c} "C=continue"
"There is not enough room on your disk
to run part one of this lesson."
GOTO "teach play"

TAG,"to play"
MESSAGE_OFF

COUNT EQUAL 9 0
IF TRUE GOTO "teach play"

RESTORE_FILE 1 "dmcorkbd"

TAG "teach play"
CALL "post_tut" IN_FILE "intro3.evn"
GOTO "menu"

TAG "end lesson"

TAG "end"
F2 {R} RETURN KEY {play.pdm} RETURN_KEY
PICTURE_ON 0 U 0 "FINDTUT.FIG"
RESTORE_FILE 0 "desktopd.cfg"
RESTORE_DT_CFG

```


INTRO2.DOC

```
TAG "Introduction 2"
M PROMPT ORG 2000-4500
ON_QUIT_GOTO "return_to_main"

PICTURE ON 1 1900 1540 "help2.fig"
EXPECT KEY {c}
PICTURE_OFF

PICTURE ON 1 1900 1540 "compass.fig"
EXPECT KEY {c}
PICTURE_OFF

POINT TO 1000 100 1 0 /* point to F1=Help */
PROMPT 1 2000,3000,4000,220 F1 "press F1"
"Let's try using DeskMate On-Line Help."
MESSAGE_OFF PICTURE_OFF

/* MESSAGE BUFFER 4200 1320 */
MESSAGE_BUFFER 4200 2500
IF TRUE_GOTO "nuf mem"
PROMPT 1 1500,1540,5000,880 {c} "C=continue"
"There is not enough memory to run this lesson"
GOTO "return_to_main"
TAG "nuf mem"
PASS_KEY

ON_QUIT_GOTO "in_help"
PAUSE MODE 100
POINT TO 1000 3400 1 3 /* info.box */
PROMPT 1 1500,1540,5000,880 {c} "C=continue"
"Help is displayed in three types of boxes. One form of Help is an
\Clinformation box\C0. An information box provides detailed
information about a specific topic.
"
PICTURE_OFF

PROMPT 1 1900,1540,4200,1100 {c} "C=continue"
"Help is \C1context sensitive\C0. This means that no matter where you
are in DeskMate -- in an application or in an accessory --
DeskMate provides \C3specific\C0 advice for the task at hand.
"

POINT TO 1600 800 1 0 /* point to text appl box */
PROMPT 1 1900,1540,4200,1100 {c} "C=continue"
"In this case, the Text application box was highlighted.
Therefore, the information box contains specific instructions
pertaining to the Text application.
"

POINT TO 3000 5050 1 0 /* point to arrows=scroll */
PROMPT 1 1900,1540,4200,660 {c} "C=continue"
"When an information box contains more text than it can
display, the \8\9 and \A\B keys may be used to scroll the text.
"
PICTURE_OFF

PROMPT 1 1900,1540,4000,660 PGDN KEY "press PAGE DOWN"
"The PAGE UP and PAGE DOWN keys can also be used to scroll
thru the text. Try pressing PAGE DOWN.
"
PASS_KEY
```



```

POINT TO 250 4850 1 2 /* cursor */
PROMPT 1 1900,1540,4200,440 PGDN KEY "press PAGE DOWN"
"The cursor moved to the bottom line. Try pressing PAGE DOWN again."
PASS KEY
PICTURE_OFF

PROMPT 1 1900,1540,4200,880 PGDN KEY "press PAGE DOWN"
"The cursor is still on the bottom line, but the text
  scrolled up one page. When you finish reading the page,
  press PAGE DOWN one more time."
PASS_KEY

POINT TO 1600 3950 1 0 /* point to [END] */
PROMPT 1 1900,1540,4200,660 {c} "C=continue"
"The word [END] has appeared, indicating that this is the last
  line of the Help information."

POINT TO 700 4860 1 3 /* point to f1=more Help */
PROMPT 1 1900,1540,4200,880 F1 "press F1"
"After reading the last line, you can either exit Help using ESC
  or get more general help on the Desktop. To get additional help,
  press F1."
PICTURE OFF MESSAGE_OFF
PASS_KEY

POINT TO 4000 1000 1 0 /* point to dotted box */
PROMPT 1 1900,2000,4200,1320 {c} "C=continue"
"General Help appears in the form of \Cltask/topic boxes\C0.
  Each box contains a question you might have about the
  current DeskMate application. The dotted box is the cursor;
  it is used to select the question you wish to have answered."
PICTURE_OFF MESSAGE_OFF

POINT TO 4000 2200 1 0 /* question box */
PROMPT 1 1900,4000,5000,660 TAB KEY "press TAB"
"Let's find the answer to the question, \"What is an
  application?\" Move the cursor to the question using
  the TAB key."
PICTURE OFF
PASS_KEY

POINT TO 1000 4850 1 3 /* enter=more details */
PROMPT 1 3000,3100,4100,1320 RETURN KEY "press ENTER"
"      And The Answer Is?\r\n\r\n
Pressing ENTER will either provide an answer to the question
  or it will provide more specific questions on the topic."
PICTURE OFF MESSAGE_OFF
PASS_KEY

POINT TO 1000 3400 1 3 /* info.box */
ESC FLAG 1
PROMPT 1 4500,1000,2200,1980 ESC KEY "press ESC"
"In this case, an information box appeared with an answer
  to the question. After reading the answer, press ESC to
  return to the list of questions."
PICTURE_OFF MESSAGE_OFF

```


PASS KEY
ESC_FLAG 0

POINT TO 4000 3300 1 0 /* question box */
PROMPT 1 1900,1540,4200,880 DOWN_ARROW "move cursor"
"Since our thirst for knowledge knows no bounds,
let's seek out the answer to the next question.
Another way to move the cursor is with \A\B.
"

PICTURE OFF MESSAGE_OFF
PASS_KEY

PICTURE ON 1 1900 1540 "magnify.fig"
EXPECT KEY RETURN KEY
PICTURE OFF MESSAGE_OFF
PASS_KEY

PROMPT 1 4500,1540,2200,880 {c} "C=continue"
"Are you wondering why an information box did not
appear with an answer?
"

PROMPT 1 4500,1000,2200,2200 RETURN KEY "press ENTER"
"Well, Help sometimes provides a more detailed response
by asking more specific questions. Press ENTER again
to see if you will get an answer to the currently
selected question.
"

PASS_KEY

PROMPT 1 4500,1000,2200,1760 {c} "C=continue"
"Eureka! You've discovered the third and final type
of Help. Instead of an information box, the answer
is being displayed in step-by-step boxes.
"

POINT TO 1500 100 1 1 /* top line */
PROMPT 1 4500,1540,2200,880 {c} "C=continue"
"The question being answered is displayed at the top of
the screen."
PICTURE_OFF

POINT TO 550 800 1 3 /* number of steps */
PROMPT 1 4500,1000,2200,1760 {c} "C=continue"
"Each step is numbered in the order that they are to
be completed. You will need to exit Help before you
can perform the instructions.
"

MESSAGE OFF
PICTURE_OFF

PICTURE ON 1 1900 1540 "powerful.fig"
EXPECT KEY RETURN KEY
PICTURE OFF MESSAGE_OFF
PASS_KEY

ESC_FLAG 1
POINT TO 1000 3400 1 3 /* info.box */
PROMPT 1 4500,1540,2200,1100 ESC_KEY "press ESC"
"The information box tells you how to perform the
first step - Choosing an application box.
"

PICTURE OFF
PASS_KEY

ESC_FLAG 0

PROMPT 1 4000,1540,3800,1100 HOME KEY "press HOME"
" There's no place like Home.\r\n\r\n
The HOME key will always take you back to the first
set of questions. Try it.
"

MESSAGE OFF PICTURE_OFF
PASS_KEY

POINT TO 4700 4000 1 3 /* f1=more Help */
PROMPT 1 1900,1540,4200,660 F1 "press F1"
"If you do not see a question or topic of interest,
you can always press F1 to see additional task /
topic boxes.
"

MESSAGE OFF PICTURE_OFF
PASS_KEY

ESC_FLAG 1
PROMPT 1 1500,4000,5500,880 ESC KEY "press ESC"
"Pressing F1 while in Help will give you additional
help. In this case, you were given six new topics
to choose from. Let's exit Help and return to the Desktop.
"

MESSAGE OFF PICTURE_OFF
PASS_KEY
ESC_FLAG 0

ON QUIT GOTO "return to main"
POINT TO 7500 370 1 T /* menu bar f10 */
PROMPT 1 1900,1540,4200,880 F10 "pull down menu"
"Let's look at how useful Help can be to explore DeskMate.
Let's say you are curious about the F10 menu. To find
out what it does, press F10.
"

MESSAGE OFF PICTURE_OFF
PASS_KEY

ON QUIT GOTO "kill menu"
POINT TO 1000 100 T 0 /* f1=Help on top line */
PROMPT 1 2500,1540,3000,660 F1 "use Help"
"Now, use Help to find out what the options in the
F10 menu are.
"

MESSAGE OFF PICTURE_OFF

MESSAGE_BUFFER 4300 440
PASS_KEY

ESC_FLAG 1
POINT TO 500 3400 1 3 /* info.box */
PROMPT 1 1900,1540,4300,440 ESC KEY "press ESC"
"Help informs you that this is The Accessories Menu.
Press ESC to exit Help.
"

MESSAGE OFF PICTURE_OFF
PASS_KEY
ESC_FLAG 0
F10
{p}


```

POINT TO 6000 1670 1 1 /* phone list option */
PROMPT 1 500,1540,4200,1100 F1 "use Help"
"Now, let's see what kind of help is available for
an individual menu option. We have already moved the
highlight to the PHONE LIST option, all you need to
do is use Help.
"
MESSAGE OFF PICTURE_OFF
PASS_KEY

ON QUIT GOTO "return to main"
POINT TO 500 3400 1 3 /* info.box */
ESC_FLAG 1
PROMPT 1 1900,1540,4100,880 ESC KEY "press ESC"
"As you can see, Help provided you with more detailed
information on the Phone List accessory. Now, let's
review what you have learned.
"
PICTURE OFF MESSAGE_OFF
PASS_KEY
ESC_FLAG 0

PICTURE ON 1 1500 1200 "superfl.fig"
EXPECT KEY {c}
GOTO "Return_to_main"

TAG "in help"
ESC_KEY

TAG "return to main"
GOTO "menu" IN_FILE "dmintro.evn"

TAG "kill menu"
GOTO "return_to_main"

```


INTRO3.DOC

```
/*-----*/
/* Introductory Lesson   - Third Part          */
/*-----*/
TAG,"Introduction_3"

ON QUIT GOTO "ea"
M_PROMPT_ORG 2200 2460

PROMPT 1 1900 1980 4200 1540 {c} "C=continue"
"          Part 3: Write a Note\r\n
\r\n
In this part we will write a short
note using the Text application, and
practice some of the things that you
have learned in the previous parts."

PROMPT 1 1500 2200 5000 440 RETURN KEY "run TEXT box"
"The highlight is already on the TEXT box, so it is
ready to be run."

MESSAGE_OFF

/* Setup files needed for part   */
PICTURE ON 0 0 0 "findtut.fig"
DISK SPACE 2048
IF_FALSE_GOTO "no_space"

TAG "get withit"
RETURN_KEY

ON_QUIT_GOTO "eb"

M PROMPT ORG 500 1540
PROMPT 1-400 1320 5200 1320 ABORT_KEY "type \"Hi Robert\""
"Type the greeting for the
note. The exact words that we want you to type are
in the lower-right corner of this box enclosed in
quotes. Type the words exactly as they appear
there, but do not type the quotes. Remember, you
can correct mistakes with BACKSPACE."

M PROMPT ORG,200,1320
SPACE KEY BKSPACE KEY
ON_QUIT_GOTO "ee"

GET_TEXT 0 "Hi Robert"

END_KEY,LEFT_ARROW

M PROMPT ORG 500 1760
PROMPT 1-400,1540,4700,660 ABORT KEY "erase \"Robert\""
"Very good. But let's make the greeting less
formal. Use BACKSPACE to erase Robert's name
so that you can change it to Bob."

COUNT_SET,0,6

TAG,"erase loop"
EXPECT KEY BKSPACE KEY
IF_TRUE_GOTO "erase_do"
```



```

BKSPACE KEY
PROMPT I 400 1540 3200 220 ABORT_KEY "erase \"Robert\""
"Keep pressing BACKSPACE."
LOOP_TO 0 "erase_loop"

TAG "erase do"
BKSPACE_KEY

LOOP_TO 0 "erase_loop"

TAG "zzapped"

PROMPT 1 400 1540 3000 220 ABORT_KEY "type \"Bob\""
"Now, type Robert's nickname."

GET TEXT 0 "Bob"
END_KEY

/* Autotype the remainder of the message */

M PROMPT ORG 2200 2420
PROMPT 1 2000 2200 4000 440 {c} "C=continue"
"Great! Now let us type the rest of
the note for you."

BKSPACE_KEY {,}

MESSAGE OFF
KEY INTERVAL 2
RETURN_KEY RETURN_KEY

MESSAGE OFF
PICTURE_OFF

{I just got my new Tandy Computer and it's terrific.
Right now I am }
{learning how to use DeskMate. }
{With it, I can do many home and office tasks }
{without having to buy extra software. }
{Such a deal!}
RETURN_KEY RETURN_KEY

MESSAGE OFF
TAB_KEY TAB_KEY TAB_KEY TAB_KEY
TAB_KEY TAB_KEY TAB_KEY
{Come and see it!}
KEY_INTERVAL 0

M PROMPT ORG 2200 3960
PROMPT 1 2000 3960 4000 880 ABORT_KEY "choose menu option"
"Now that we have completed the note
you need to save it on the disk
before you exit Text. Choose SAVE AS
from the File Menu."

GET TO MENU 2 4 "File" "SAVE AS" SPACE_KEY
ON_QUIT_GOTO "ee"

```



```

M PROMPT ORG 2200 3960
PROMPT 1-1900 3740 4200 880 ABORT KEY "type \"BOB\""
"The first step is to type the name of
the file to save the note to at the
Save as: prompt. The cursor is already
there, so type the name of the file."

M PROMPT ORG 2000 3300
GET_TEXT 0 "Bob"

M PROMPT ORG 2200 3740
PROMPT 1-2200 3740 3600 440 TAB KEY "move cursor"
"Next, press TAB to move the cursor
to the SAVE button."
PASS_KEY

PICTURE_ON 0 0 0 "findtut.fig"

COUNT SET 7 2
FILE EXIST 0 "bob.doc"
IF_FALSE_GOTO "push ok"

PRESERVE FILE 0 "bob.doc"
DELETE FILE 0 "bob.$b$"
COUNT SET 7 1

ON_QUIT_GOTO "remove_bob"

TAG "push ok"
PROMPT 1 1700 3740 4800 220 SPACE KEY "push button"
"And finally, press space bar to save the note."
PASS_KEY

M PROMPT ORG 2000 3960
PROMPT 1-1200 3960 5600 880 ABORT KEY "choose menu option"
"Your note has been saved to disk. If there had been a
problem with the saving of the file DeskMate would
have notified you. Let's exit Text and go back to the
Desktop. Choose EXIT from the File Menu."

GET TO MENU 2 9 "File" "EXIT" SPACE_KEY
ON_QUIT_GOTO "ea"

MESSAGE_OFF

LOOP TO 7 "delete saved"
RESTORE FILE 0 "bob.doc"
GOTO "DONE"

TAG "delete saved"
DELETE_FILE 0 "bob.doc"

TAG "done"

ON_QUIT_GOTO "eb"

PICTURE ON 0 110 880 "desktxt.fig"
POINT_TO 1400, 1100, 1, 0 /* Point to the left */

M PROMPT ORG 2200 2420
PROMPT 1-2000 2200 4000 440 {c} "C=continue"
"Notice that your note now appears in
the TEXT box as BOB.DOC."

```



```

PICTURE_OFF

PROMPT 1 1900 2200 5400 1100 {c} "C=lesson menu"
"In part three of this lesson, we have reviewed:\r\n
\r\n
- How to use dialog boxes.\r\n
- How to choose an option from a menu."

F2 {r} RETURN_KEY ESC_KEY
GOTO "ea"

TAG "no_extra"
RETURN_KEY

/* problems with lesson */

TAG "no_space"

PROMPT 1 880 2000 4100 440 {c} "C=lesson menu"
"There is not enough room on your disk
to run part three of this lesson."

GOTO "exit"

TAG "remove bob"
MESSAGE OFF
ESC_KEY
IN MSGBOX "Save Changes"
IF FALSE GOTO "del_bob"
ALT_N

TAG "del bob"
LOOP TO 7 "delete saved q"
RESTORE FILE 0 "bob.doc"
GOTO "ea"

TAG "delete saved q"
DELETE FILE 0 "bob.doc"
GOTO "ea"

TAG "ef"
MESSAGE OFF
ESC_KEY

TAG "ee"
MESSAGE OFF
ESC_KEY
ALT_N
GOTO "menu" IN_FILE "dmintro.evn"

TAG "ed"
MESSAGE OFF
ESC_KEY

TAG "ec"
MESSAGE OFF
ESC_KEY

TAG "eb"
MESSAGE OFF
ESC_KEY

TAG "ea"
MESSAGE OFF

```



```

TAG "exit"
GOTO "menu" IN_FILE "dmintro.evn"

/*-----*/
TAG "post tut"
ON_QUIT_GOTO "on_their_own"

M_PROMPT_ORG 2100 2420

POINT TO 1600 3960 1 3
PROMPT 1 2000 2200 4900 1100 TAB KEY "move highlight"
"
    Part 4: Using Teach Me!\r\n
\r\n
To run other Teach Me lessons, first move the
highlight to the Teach Me box. Use TAB to move
the highlight."

PASS_KEY

COUNT_SET 0 4

TAG "to teach me loop"
POINT TO 1600 3960 1 3
PROMPT 1 400 2640 3300 220 TAB KEY "move highlight"
"You are doing fine. Keep going."
PASS_KEY
LOOP TO 0 "to_teach_me_loop"
PICTURE_OFF

M PROMPT ORG 500 2860
PROMPT 1 400 3080 5000 220 RETURN KEY "press ENTER"
"To run the Teach Me application, press ENTER."
MESSAGE OFF
PASS_KEY

ON_QUIT_GOTO "quit play"

HOME_KEY HOME_KEY HOME_KEY HOME_KEY HOME_KEY HOME_KEY

M PROMPT ORG 2200 2420
PROMPT 1 1400 2200 4900 660 {c} "C=continue"
"Teach Me lets you run other lessons. Each
lesson will guide you through the fundamentals
of a DeskMate application."

MESSAGE_OFF

M PROMPT ORG 300 3080
POINT TO 4000 2200 1 1
PROMPT 1 200 2860 3800 1100 {c} "C=continue"
"The list box to the right contains
a list of the available lessons.
To run a lesson, use \A\B to move
the highlight to the lesson that
you want and then press ENTER."
PICTURE OFF
MESSAGE_OFF

```



```

M PROMPT ORG 300 2200
POINT TO 4100 1320 1 1
PROMPT 1 200 1980 4100 1100 {c} "C=continue"
"\C1DeskMate: An Introduction\C0 is the
lesson that you are currently running.
Feel free to run it again or to run
any of the others that you see listed
after you have completed this lesson."
PICTURE OFF
MESSAGE OFF

ESC FLAG 1
M PROMPT ORG 2200 2640
PROMPT 1 1400 2420 4000 660 ESC KEY "press ESC"
"To exit Teach Me without running a
lesson, just press ESC. Try it now
to return to the Desktop application."
MESSAGE OFF
PASS KEY
ESC_FLAG 0

ON_QUIT_GOTO "on_their_own"

PROMPT 1 1600 2200 4700 660 {c} "C=lesson menu"
"You can use Teach Me to become familiar with
a new DeskMate application and to get ideas
about what you can do with that application."

GOTO "on_their_own"

TAG "quit play"
MESSAGE OFF PICTURE OFF
ESC_KEY

TAG "on their own"
MESSAGE OFF PICTURE OFF
RETURN

```


OPTIONS.DOC

```
TAG "options"
LEFT_ARROW      LEFT_ARROW      RIGHT_ARROW
DOWN_ARROW      LEFT_ARROW      LEFT_ARROW
UP_ARROW        CTRL_HOME_KEY   UP_ARROW

ON_QUIT_GOTO "end lesson"
M_PROMPT_ORG 1000 1760

/* MESSAGE BUFFER 6400 3190 */
MESSAGE_BUFFER 6400 2900
IF TRUE_GOTO "MEMORY OK"
PROMPT U 1300 1320 5500 1980 {c} "c=end lesson"
"
    Not Enough Memory\r\n
Your computer does not have enough memory to complete
this lesson. You may be able to complete the lesson
using a lower resolution video mode (read dmvid.doc
to change the video mode). You can also save memory
by eliminating any programs that terminate, but stay
resident in memory. Also make sure that you do not
use the TASKSWITCH option from the Accessories Menu
before taking a lesson.
"
GOTO "end lesson"

TAG "MEMORY OK"
PROMPT 1 90U,1540,6200,2420 {c} "C=continue"
"
    Teach Me About DeskMate\r\n
\r\n
This lesson will introduce you to the fundamentals of
DeskMate and give you a sampling of the things you can
do with it.\r\n
\r\n
First, let's see how to use this lesson. The lower right
corner of each instruction box (like this one) will tell you
how to get to the next step of the lesson. In this case,
the C=continue message means that you should press C on
your keyboard to continue. Please press it now."

M_PROMPT_ORG 1500 1760
PROMPT 1 1400 1540 5300 1760 {c} "C=continue"
"If you are new to DeskMate or computers, you are
going to make some mistakes. Don't worry about it,
making mistakes is an important part of learning.
In particular, don't worry about pressing the
wrong key. We will catch it and tell you what you
need to do next. If you are ever unsure of what
to do, try something and we will guide you through.
You can press the ESC key to quit during a lesson."

MESSAGE_OFF

TAG "show menu"
ON_QUIT_GOTO "end lesson"

CTRL_U
LEFT_ARROW      LEFT_ARROW      RIGHT_ARROW
DOWN_ARROW      LEFT_ARROW      LEFT_ARROW
UP_ARROW        CTRL_HOME_KEY   UP_ARROW

MOUSE_ON
IF_FALSE_GOTO "no mouse opt"
MOUSE_OFF
```



```

OPTIONS 1 6 30 2400 1320
"    Teach Me About DeskMate"
"1. Desktop Basics"          "Section 1"
"2. Use Help"                 "Section 2"
"3. Write a Note"             "Section 3"
"4. Run Other Lessons"        "to play"
"5. Use a Mouse"              "A"
"6. Exit This Lesson"         "end lesson"

TAG "no mouse opt"
OPTIONS 1 5 30 2400 1320
"    Teach Me About DeskMate"
"1. Desktop Basics"          "Section 1"
"2. Use Help"                 "Section 2"
"3. Write a Note"             "Section 3"
"4. Run Other Lessons"        "to play"
"5. Exit This Lesson"         "end lesson"

TAG "Section 1"
MESSAGE OFF PICTURE OFF
GOTO "Section_1" IN_FILE "dmintro.evn"

TAG "Section 2"
MESSAGE OFF PICTURE OFF
GOTO "Introduction_2" IN_FILE "intro2.evn"

TAG "Section 3"
MESSAGE OFF PICTURE OFF
GOTO "Introduction_3" IN_FILE "intro3.evn"

TAG "to play"
MESSAGE OFF PICTURE OFF
GOTO "to_play" IN_FILE "dmintro.evn"

TAG "A"
MESSAGE OFF PICTURE OFF
GOTO "A" IN_FILE "mouse.evn"

TAG "end lesson"
MESSAGE OFF PICTURE OFF
GOTO "end lesson" IN_FILE "dmintro.evn"

```


MOUSE.DOC

```

/*-----* Mouse Lesson *-----*/
/*-----*-----*/

TAG "A"
M_PROMPT_ORG 2700 3300

MOUSE_ON

ON_QUIT_GOTO "QQ1"

PICTURE ON 0 1000 440 "mouse.fig"
EXPECT_KEY {c}

PICTURE ON 0 1000 440 "rat.fig"
EXPECT_KEY {c}

PICTURE ON 0 1000 440 "arrow.fig"
EXPECT_KEY {c}

F2 {R} RETURN_KEY {hangman} RETURN_KEY
CTRL_T

ON_QUIT_GOTO "Q"

PAUSE_MODE 10

IN MSGBOX "Hangman"
IF FALSE GOTO "Mc1"
TAB_KEY SPACE_KEY

TAG "Mc1"
TAB_KEY SPACE_KEY SPACE_KEY SPACE_KEY
TAB_KEY TAB_KEY TAB_KEY
POINT_TO 3U00, 385U, 1, 1 /* 1 word per game */
/* point to the right*/

PROMPT 1 800, 440, 6500, 1100 ABORT_KEY "push OK"
"A mouse gives DeskMate and other popular business, educational,
and entertainment programs \"point and click\" convenience. To
see how, let's use the mouse to play Hangman. To start the
game, move the mouse pointer to point to the OK button and
\"push\" it by clicking one of the mouse's buttons."

COUNT SET 1 3000, COUNT SET 2 3740, COUNT_SET 3 900, COUNT_SET 4 440
GET_MOUSE 0 MS_CLICK 1 2 3 4
PICTURE OFF
MESSAGE OFF
IF FALSE GOTO "CL1"
PASS_MOUSE
GOTO "C2"

TAG "CL1"
WAS_MOUSE 0 MS_BUTTON_DOWN 1 2 3 4
IF TRUE GOTO "E10"
WAS_MOUSE 0 MS_DBL_CLICK 1 2 3 4
IF TRUE GOTO "E11"
WAS_MOUSE 0 MS_SHFT_CLICK 1 2 3 4
IF TRUE GOTO "E12"
WAS_MOUSE 0 MS_HOLD 1 2 3 4
IF TRUE GOTO "AGAIN1"
WAS_MOUSE 0 MS_BUTTON_UP 1 2 3 4
IF TRUE GOTO "AGAIN1"
```



```

GOTO"E13"

TAG"E10"
PROMPT 1 1500 4180 5000 660 ABORT KEY "click mouse"
"To click the mouse, briefly tap the mouse
button; do not hold the button down. Try again
to click on OK."
GOTO"AGAIN1"

TAG"E11"
PROMPT 1 1500 4180 5000 440 ABORT KEY "click mouse"
"Do not double-click. Just briefly tap the mouse
button once. Try again to click on OK."
GOTO"AGAIN1"

TAG"E12"
PROMPT 1 1500 4180 5000 440 ABORT KEY "click mouse"
"Do not press the SHIFT key when clicking. Try
again to click on OK."
GOTO"AGAIN1"

TAG"E13"
PROMPT 1 1500 4180 5000 660 ABORT KEY "click mouse"
"Before clicking the mouse, position the mouse
pointer so that it points to the center of the
button. Try again to click on OK."

TAG"AGAIN1"
GET MOUSE 0 MS CLICK 1 2 3 4
IF_FALSE_GOTO"CL1"

MESSAGE OFF
PASS_MOUSE

/*-----*/
TAG"C2"
PROMPT 1 2000, 4290, 4000, 220 ABORT_KEY "type your name"
"Type in your name and press enter."

COUNT SET 1 12
PASS WHILE 1 {A} .. {z} RIGHT_ARROW LEFT_ARROW BKSPACE_KEY DELETE_KEY []
MESSAGE OFF
RETURN_KEY

/*-----*/
CTRL_W {mouse} RETURN_KEY

ON QUIT GOTO "Q2"
POINT_TO 1450, 4510, 1, 1 /* Point to the right */

```



```

PROMPT 1 200, 3080, 2500, 660 ABORT_KEY "click on \"A\""
"Let's try to guess the
word. To start, click
on the letter \"A\"."

COUNT_SET 1 1500, COUNT_SET 2 4400, COUNT_SET 3 200, COUNT_SET 4 320
GET_MOUSE 1 MS_CLICK 1 2 3 4
MESSAGE OFF
PICTURE OFF
IF_TRUE_GOTO "C3"

TAG"CL2"
WAS_MOUSE 0 MS_BUTTON_DOWN 1 2 3 4
IF_TRUE_GOTO "E20"
WAS_MOUSE 0 MS_DBL_CLICK 1 2 3 4
IF_TRUE_GOTO "E21"
WAS_MOUSE 0 MS_SHFT_CLICK 1 2 3 4
IF_TRUE_GOTO "E22"
WAS_MOUSE 0 MS_HOLD 1 2 3 4
IF_TRUE_GOTO "AGAIN2"
WAS_MOUSE 0 MS_BUTTON_UP 1 2 3 4
IF_TRUE_GOTO "AGAIN2"
GOTO"E23"

TAG"E20"
PROMPT 1 1500 2200 5000 660 ABORT_KEY "click mouse"
"To click the mouse, briefly tap the mouse
button; do not hold the button down. Try again
to click on \"A\"."
GOTO"AGAIN2"

TAG"E21"
PROMPT 1 1500 2200 5000 440 ABORT_KEY "click mouse"
"Do not double-click. Just briefly tap the mouse
button once. Try again to click on \"A\"."
GOTO"AGAIN2"

TAG"E22"
PROMPT 1 1500 2200 5000 440 ABORT_KEY "click mouse"
"Do not press the SHIFT key when clicking. Try
again to click on \"A\"."
GOTO"AGAIN2"

TAG"E23"
PROMPT 1 1500 2200 5000 660 ABORT_KEY "click mouse"
"Before clicking the mouse, position the mouse
pointer so that it points to the center of the
button. Try again to click on \"A\"."

TAG"AGAIN2"
GET_MOUSE 0 MS_CLICK 1 2 3 4
IF_FALSE_GOTO"CL2"

MESSAGE OFF
PASS_MOUSE

/*-----*/
TAG"C3"
PROMPT 1 200, 3080, 2500, 660 ABORT_KEY "click on \"E\""
"Try the letter \"E\"
next. Click on the \"E\"
button."

```



```
COUNT SET 1 3050, COUNT SET 2 4400, COUNT_SET 3 200, COUNT_SET 4 320
GET MOUSE 1 MS_CLICK 1 2 3 4
MESSAGE OFF
IF_TRUE_CALL "C4"
```

```
TAG"CL3"
WAS MOUSE 0 MS BUTTON_DOWN 1 2 3 4
IF TRUE GOTO "E30"
WAS MOUSE 0 MS DBL_CLICK 1 2 3 4
IF TRUE GOTO "E31"
WAS MOUSE 0 MS SHFT_CLICK 1 2 3 4
IF TRUE GOTO "E32"
WAS MOUSE 0 MS HOLD 1 2 3 4
IF TRUE GOTO "AGAIN3"
WAS MOUSE 0 MS BUTTON_UP 1 2 3 4
IF TRUE GOTO "AGAIN3"
GOTO"E33"
```

```
TAG"E30"
PROMPT 1 1500 2200 5000 660 ABORT KEY "click mouse"
"To click the mouse, briefly tap the mouse
button; do not hold the button down. Try again
to click on \"E\"."
GOTO"AGAIN3"
```

```
TAG"E31"
PROMPT 1 1500 2200 5000 440 ABORT KEY "click mouse"
"Do not double-click. Just briefly tap the mouse
button once. Try again to click on \"E\"."
GOTO"AGAIN3"
```

```
TAG"E32"
PROMPT 1 1500 2200 5000 440 ABORT KEY "click mouse"
"Do not press the SHIFT key when clicking. Try
again to click on \"E\"."
GOTO"AGAIN3"
```

```
TAG"E33"
PROMPT 1 1500 2200 5000 660 ABORT KEY "click mouse"
"Before clicking the mouse, position the mouse
pointer so that it points to the center of the
button. Try again to click on \"E\"."
```

```
TAG"AGAIN3"
GET MOUSE 0 MS_CLICK 1 2 3 4
IF_FALSE_GOTO"CL3"
```

```
MESSAGE OFF
PASS_MOUSE
```

```
/*-----*/
```

```
TAG"C4"
PROMPT 1 200, 3080, 2500, 220 ABORT_KEY "click on \"I\""
"Try the letter \"I\"."
```

```
COUNT SET 1 4650, COUNT SET 2 4400, COUNT_SET 3 200, COUNT_SET 4 320
GET MOUSE 1 MS_CLICK 1 2 3 4
MESSAGE OFF
IF_TRUE_GOTO "C5"
```



```

TAG"CL4"
WAS MOUSE 0 MS BUTTON_DOWN 1 2 3 4
IF TRUE GOTO "E40"
WAS MOUSE 0 MS DBL_CLICK 1 2 3 4
IF TRUE GOTO "E41"
WAS MOUSE 0 MS SHFT_CLICK 1 2 3 4
IF TRUE GOTO "E42"
WAS MOUSE 0 MS HOLD 1 2 3 4
IF TRUE GOTO "AGAIN4"
WAS MOUSE 0 MS BUTTON_UP 1 2 3 4
IF TRUE GOTO "AGAIN4"
GOTO"E43"

TAG"E40"
PROMPT 1 1500 2200 5000 660 ABORT KEY "click mouse"
"To click the mouse, briefly tap the mouse
button; do not hold the button down. Try again
to click on \"I\"."
GOTO"AGAIN4"

TAG"E41"
PROMPT 1 1500 2200 5000 440 ABORT KEY "click mouse"
"Do not double-click. Just briefly tap the mouse
button once. Try again to click on \"I\"."
GOTO"AGAIN4"

TAG"E42"
PROMPT 1 1500 2200 5000 440 ABORT KEY "click mouse"
"Do not press the SHIFT key when clicking. Try
again to click on \"I\"."
GOTO"AGAIN4"

TAG"E43"
PROMPT 1 1500 2200 5000 660 ABORT KEY "click mouse"
"Before clicking the mouse, position the mouse
pointer so that it points to the center of the
button. Try again to click on \"I\"."

TAG"AGAIN4"
GET MOUSE 0 MS CLICK 1 2 3 4
IF_FALSE_GOTO"CL4"

MESSAGE OFF
PASS_MOUSE

/*-----*/
TAG"C5"
PROMPT 1 200, 3080, 2500, 220 ABORT_KEY "click on \"O\""
"Try the letter \"O\"."

COUNT SET 1 1850, COUNT SET 2 4840, COUNT_SET 3 200, COUNT_SET 4 320
GET MOUSE 1 MS_CLICK 1 2 3 4
MESSAGE OFF
IF_TRUE_GOTO "C6"

TAG"CL5"
WAS MOUSE 0 MS BUTTON_DOWN 1 2 3 4
IF TRUE GOTO "E50"
WAS MOUSE 0 MS DBL_CLICK 1 2 3 4
IF TRUE GOTO "E51"
WAS MOUSE 0 MS SHFT_CLICK 1 2 3 4
IF TRUE GOTO "E52"
WAS MOUSE 0 MS HOLD 1 2 3 4
IF_TRUE_GOTO "AGAIN5"

```



```
WAS MOUSE 0 MS BUTTON UP 1 2 3 4
IF TRUE GOTO "AGAIN5"
GOTO"E53"
```

```
TAG"E50"
PROMPT 1 1500 2200 5000 660 ABORT KEY "click mouse"
"To click the mouse, briefly tap the mouse
button; do not hold the button down. Try again
to click on \"0\"."
GOTO"AGAIN5"
```

```
TAG"E51"
PROMPT 1 1500 2200 5000 440 ABORT KEY "click mouse"
"Do not double-click. Just briefly tap the mouse
button once. Try again to click on \"0\"."
GOTO"AGAIN5"
```

```
TAG"E52"
PROMPT 1 1500 2200 5000 440 ABORT KEY "click mouse"
"Do not press the SHIFT key when clicking. Try
again to click on \"0\"."
GOTO"AGAIN5"
```

```
TAG"E53"
PROMPT 1 1500 2200 5000 660 ABORT KEY "click mouse"
"Before clicking the mouse, position the mouse
pointer so that it points to the center of the
button. Try again to click on \"0\"."
```

```
TAG"AGAIN5"
GET MOUSE 0 MS CLICK 1 2 3 4
IF_FALSE_GOTO"CL5"
```

```
MESSAGE OFF
PASS_MOUSE
```

```
/*-----*/
TAG"C6"
PROMPT 1 200, 3080, 2500, 220 ABORT_KEY "click on \"U\"""
"Try the letter \"U\"."
```

```
COUNT SET 1 4250, COUNT SET 2 4840, COUNT_SET 3 200, COUNT_SET 4 320
GET MOUSE 1 MS_CLICK 1 2 3 4
MESSAGE OFF
IF_TRUE_GOTO "C7"
```

```
TAG"CL6"
WAS MOUSE 0 MS BUTTON_DOWN 1 2 3 4
IF TRUE GOTO "E60"
WAS MOUSE 0 MS DBL_CLICK 1 2 3 4
IF TRUE GOTO "E61"
WAS MOUSE 0 MS SHFT_CLICK 1 2 3 4
IF TRUE GOTO "E62"
WAS MOUSE 0 MS HOLD 1 2 3 4
IF TRUE GOTO "AGAIN6"
WAS MOUSE 0 MS BUTTON UP 1 2 3 4
IF TRUE GOTO "AGAIN6"
GOTO"E63"
```

```
TAG"E60"
PROMPT 1 1500 2200 5000 660 ABORT KEY "click mouse"
"To click the mouse, briefly tap the mouse
button; do not hold the button down. Try again
to click on \"U\"."
```


GOTO"AGAIN6"

TAG"E61"

PROMPT 1 1500 2200 5000 440 ABORT KEY "click mouse"
"Do not double-click. Just briefly tap the mouse
button once. Try again to click on \"U\"."
GOTO"AGAIN6"

TAG"E62"

PROMPT 1 1500 2200 5000 440 ABORT KEY "click mouse"
"Do not press the SHIFT key when clicking. Try
again to click on \"U\"."
GOTO"AGAIN6"

TAG"E63"

PROMPT 1 1500 2200 5000 660 ABORT KEY "click mouse"
"Before clicking the mouse, position the mouse
pointer so that it points to the center of the
button. Try again to click on \"U\"."

TAG"AGAIN6"

GET MOUSE 0 MS CLICK 1 2 3 4
IF_FALSE_GOTO"CL6"

MESSAGE OFF
PASS_MOUSE

/*-----*/

TAG"C7"

PROMPT 1 200, 3080, 2500, 220 ABORT_KEY "click on \"M\""
"Try the letter \"M\"."

COUNT SET 1 6300, COUNT SET 2 4400, COUNT_SET 3 200, COUNT_SET 4 320
GET MOUSE 1 MS_CLICK 1 2 3 4
MESSAGE OFF
IF_TRUE_GOTO "C8"

TAG"CL7"

WAS MOUSE 0 MS BUTTON_DOWN 1 2 3 4
IF_TRUE_GOTO "E70"
WAS MOUSE 0 MS DBL_CLICK 1 2 3 4
IF_TRUE_GOTO "E71"
WAS MOUSE 0 MS SHFT_CLICK 1 2 3 4
IF_TRUE_GOTO "E72"
WAS MOUSE 0 MS HOLD 1 2 3 4
IF_TRUE_GOTO "AGAIN7"
WAS MOUSE 0 MS BUTTON_UP 1 2 3 4
IF_TRUE_GOTO "AGAIN7"
GOTO"E73"

TAG"E70"

PROMPT 1 1500 2200 5000 660 ABORT KEY "click mouse"
"To click the mouse, briefly tap the mouse
button; do not hold the button down. Try again
to click on \"M\"."
GOTO"AGAIN7"

TAG"E71"

PROMPT 1 1500 2200 5000 440 ABORT KEY "click mouse"
"Do not double-click. Just briefly tap the mouse
button once. Try again to click on \"M\"."
GOTO"AGAIN7"


```

TAG"E72"
PROMPT 1 1500 2200 5000 440 ABORT KEY "click mouse"
"Do not press the SHIFT key when clicking. Try
again to click on \"M\"."
GOTO"AGAIN7"

TAG"E73"
PROMPT 1 1500 2200 5000 660 ABORT KEY "click mouse"
"Before clicking the mouse, position the mouse
pointer so that it points to the center of the
button. Try again to click on \"M\"."

TAG"AGAIN7"
GET MOUSE 0 MS CLICK 1 2 3 4
IF_FALSE_GOTO"CL7"

MESSAGE OFF
PASS_MOUSE

/*-----*/
TAG"C8"
PROMPT 1 200, 3080, 2500, 440 ABORT_KEY "click on \"S\"""
"You just about have it;
Try the letter \"S\"."

COUNT SET 1 3450, COUNT SET 2 4840, COUNT_SET 3 200, COUNT_SET 4 320
GET MOUSE 1 MS_CLICK 1 2 3 4
MESSAGE OFF
IF_TRUE_GOTO "C9"

TAG"CL8"
WAS MOUSE 0 MS BUTTON_DOWN 1 2 3 4
IF_TRUE_GOTO "E80"
WAS MOUSE 0 MS DBL CLICK 1 2 3 4
IF_TRUE_GOTO "E81"
WAS MOUSE 0 MS SHFT CLICK 1 2 3 4
IF_TRUE_GOTO "E82"
WAS MOUSE 0 MS HOLD 1 2 3 4
IF_TRUE_GOTO "AGAIN8"
WAS MOUSE 0 MS BUTTON UP 1 2 3 4
IF_TRUE_GOTO "AGAIN8"
GOTO"E83"

TAG"E80"
PROMPT 1 1500 2200 5000 660 ABORT KEY "click mouse"
"To click the mouse, briefly tap the mouse
button; do not hold the button down. Try again
to click on \"S\"."
GOTO"AGAIN8"

TAG"E81"
PROMPT 1 1500 2200 5000 440 ABORT KEY "click mouse"
"Do not double-click. Just briefly tap the mouse
button once. Try again to click on \"S\"."
GOTO"AGAIN8"

TAG"E82"
PROMPT 1 1500 2200 5000 440 ABORT KEY "click mouse"
"Do not press the SHIFT key when clicking. Try
again to click on \"S\"."
GOTO"AGAIN8"

```



```

TAG"E83"
PROMPT 1 1500 2200 5000 660 ABORT KEY "click mouse"
"Before clicking the mouse, position the mouse
pointer so that it points to the center of the
button. Try again to click on \"S\"."

TAG"AGAIN8"
GET MOUSE 0 MS CLICK 1 2 3 4
IF_FALSE_GOTO"CL8"

MESSAGE OFF
PASS_MOUSE

/*-----*/
TAG"C9"
IN MSGBOX "Hangman"
IF_FALSE_GOTO "C9"

ON QUIT GOTO "N1"
PROMPT 1 1400, 4400, 5000, 440 ABORT KEY "push NO"
"Click on NO to exit Hangman and we'll show you
how to use the mouse in other ways."

COUNT SET 1 4500, COUNT SET 2 3190, COUNT_SET 3 700, COUNT_SET 4 330
GET MOUSE 0 MS CLICK 1 2 3 4
IF_TRUE_GOTO "CA2"

TAG"CL9"
WAS MOUSE 0 MS BUTTON_DOWN 1 2 3 4
IF_TRUE_GOTO "E90"
WAS MOUSE 0 MS DBL_CLICK 1 2 3 4
IF_TRUE_GOTO "E91"
WAS MOUSE 0 MS SHFT_CLICK 1 2 3 4
IF_TRUE_GOTO "E92"
WAS MOUSE 0 MS HOLD 1 2 3 4
IF_TRUE_GOTO "AGAIN9"
WAS MOUSE 0 MS BUTTON_UP 1 2 3 4
IF_TRUE_GOTO "AGAIN9"
GOTO"E93"

TAG"E90"
PROMPT 1 1400 4400 5000 660 ABORT KEY "click mouse"
"To click the mouse, briefly tap the mouse
button; do not hold the button down. Try again
to click on NO."
GOTO"AGAIN9"

TAG"E91"
PROMPT 1 1400 4400 5000 440 ABORT KEY "click mouse"
"Do not double-click. Just briefly tap the mouse
button once. Try again to click on NO."
GOTO"AGAIN9"

TAG"E92"
PROMPT 1 1400 4400 5000 440 ABORT KEY "click mouse"
"Do not press the SHIFT key when clicking. Try
again to click on NO."
GOTO"AGAIN9"

TAG"E93"
PROMPT 1 1400 4400 5000 660 ABORT KEY "click mouse"
"Before clicking the mouse, position the mouse
pointer so that it points to the center of the
button. Try again to click on NO."

```



```

TAG"AGAIN9"
GET MOUSE 0 MS CLICK 1 2 3 4
IF_FALSE_GOTO "CL9"

TAG"CA2"
MESSAGE OFF
PASS MOUSE
GOTO"CA" IN_FILE "MOUSE1.DOC"

/*-----*/
/*-----* Exit Tutorial (From Lesson Menu) *-----*/
/*-----*/
TAG "DE" /* EXIT APP */
PROMPT 1 1500 4180 5000 440 {c} "C=continue"
"There is not enough space on this disk space to
run this tutorial.."
GOTO "Q1"

TAG "Q3" /* EXIT APP */
MESSAGE OFF PICTURE_OFF
ESC_KEY
GOTO "Q"

TAG "Q2" /* EXIT APP */
MESSAGE OFF PICTURE_OFF
ESC_KEY

TAG "N1"
MESSAGE OFF PICTURE_OFF
ALT_N

TAG "Q" /* EXIT_TUTORIAL */
MESSAGE OFF PICTURE_OFF
MOUSE_OFF

TAG "Q1" /* EXIT */
TAG "QQ1"
MESSAGE OFF PICTURE OFF
F2 {R} RETURN_KEY ESC_KEY

TAG "END"
GOTO "menu" IN_FILE "dmintro.evn"

```


MOUSE1.DOC

```
/*-----*/
TAG"CA"
ON_QUIT_GOTO "Q" IN_FILE ""

POINT TO 1400, 770, 0, 0 /* Point to the left */
PROMPT 1 1200, 2640, 5600, 1100 ABORT KEY "double-click mouse"
"Let's go into the Text application. To run a program
from the Desktop with the mouse, you must double-click
on the application name. Position the mouse pointer on
the word \"TEXT\" and click one of the buttons
twice without pausing between clicks."

TAG"DX2"
COUNT SET 1 300, COUNT SET 2 715, COUNT_SET 3 1000, COUNT_SET 4 220
GET MOUSE 0 MS DBL_CLICK 1 2 3 4
IF_TRUE_GOTO "DX"

WAS MOUSE 0 MS CLICK 1 2 3 4
IF_TRUE_GOTO "DX1"
WAS MOUSE 0 MS SHFT_CLICK 1 2 3 4
IF_TRUE_GOTO "DX1"
WAS MOUSE 0 MS BUTTON_DOWN 1 2 3 4
IF_TRUE_GOTO "DX1"
WAS MOUSE 0 MS BUTTON_UP 1 2 3 4
IF_TRUE_GOTO "DX1"
WAS MOUSE 0 MS HOLD 1 2 3 4
IF_TRUE_GOTO "DX1"

PROMPT 1 1500 4180 5000 440 ABORT KEY "double-click mouse"
"Position the mouse on the word \"TEXT\" before
double-clicking."
GOTO "DX2"

TAG"DX1"
PROMPT 1 1500 4180 5000 880 ABORT KEY "double-click mouse"
"To double-click the mouse, briefly tap the mouse
button twice; do not pause between clicks and do
not hold the mouse button down. Try again to
double-click on \"TEXT\"."
GOTO "DX2"

TAG "DX"
MESSAGE OFF
PASS_MOUSE

ON_QUIT_GOTO "Q3" IN_FILE "MOUSE.DOC"
CTRL_HOME_KEY

/*-----*/
POINT TO 500, 660, 1, 2 /* Point to the right */
PROMPT 1 2650, 3300, 5200, 880 ABORT KEY "click on file menu"
"Let's change the document we created earlier in
Text. Retrieve this file using the OPEN option
from the File Menu. To choose the menu option with
a mouse, first, click on the File Menu."

COUNT SET 1 100, COUNT SET 2 330, COUNT_SET 3 600, COUNT_SET 4 220
GET MOUSE 0 MS CLICK 1 2 3 4
IF_TRUE_GOTO "CD"
```



```

TAG"CL9"
WAS MOUSE 0 MS BUTTON_DOWN 1 2 3 4
IF TRUE GOTO "E90"
WAS MOUSE 0 MS DBL CLICK 1 2 3 4
IF TRUE GOTO "E91"
WAS MOUSE 0 MS SHFT_CLICK 1 2 3 4
IF TRUE GOTO "E92"
WAS MOUSE 0 MS HOLD 1 2 3 4
IF TRUE GOTO "AGAIN9"
WAS MOUSE 0 MS BUTTON_UP 1 2 3 4
IF TRUE GOTO "AGAIN9"
GOTO"E93"

TAG"E90"
PROMPT 1 1400 4400 5000 660 ABORT KEY "click mouse"
"To click the mouse, briefly tap the mouse
  button; do not hold the button down. Try again
  to click on the File Menu."
GOTO"AGAIN9"

TAG"E91"
PROMPT 1 1400 4400 5000 440 ABORT KEY "click mouse"
"Do not double-click. Just briefly tap the mouse
  button once. Try again to click on the File Menu."
GOTO"AGAIN9"

TAG"E92"
PROMPT 1 1400 4400 5000 440 ABORT KEY "click mouse"
"Do not press the SHIFT key when clicking. Try
  again to click on the File Menu."
GOTO"AGAIN9"

TAG"E93"
PROMPT 1 1400 4400 5000 660 ABORT KEY "click mouse"
"Before clicking the mouse, position the mouse
  pointer so that it points to the center of the
  menu. Try again to click on File Menu."

TAG"AGAIN9"
GET MOUSE 0 MS CLICK 1 2 3 4
IF_FALSE_GOTO "CL9"

TAG "CD"
MESSAGE OFF
PICTURE OFF
PASS_MOUSE

/*-----*/
PICTURE ON 0 0 0 "FINDTUT.FIG"
DISK SPACE 1024
IF_FALSE_GOTO "DE"

PROMPT 1 2650, 3300, 5200, 220 ABORT KEY "choose menu option"
"To choose the OPEN option, double-click on OPEN."

TAG"DE2"
COUNT SET 1 0, COUNT SET 2 880, COUNT_SET 3 2600, COUNT_SET 4 220
GET MOUSE 0 MS DBL_CLICK 1 2 3 4
IF_TRUE_GOTO "DJ"

WAS MOUSE 0 MS CLICK 1 2 3 4
IF TRUE GOTO "DE1"
WAS MOUSE 0 MS SHFT_CLICK 1 2 3 4
IF_TRUE_GOTO "DE1"

```



```

WAS MOUSE 0 MS BUTTON_DOWN 1 2 3 4
IF TRUE GOTO "DE1"
WAS MOUSE 0 MS BUTTON_UP 1 2 3 4
IF TRUE GOTO "DE1"
WAS MOUSE 0 MS HOLD 1 2 3 4
IF TRUE GOTO "DE1"

PROMPT 1 1500 4180 5000 440 ABORT KEY "double-click mouse"
"Position the mouse on the OPEN option before
double-clicking."
GOTO "DE2"

TAG"DE1"
PROMPT 1 1500 4180 5000 880 ABORT KEY "double-click mouse"
"To double-click the mouse, briefly tap the mouse
button twice; do not pause between clicks and do
not hold the mouse button down. Try again to
double-click on File Menu."

GOTO "DE2"

TAG "DJ"
MESSAGE OFF
PASS MOUSE
PICTURE_ON 0 2400 2255 "listbox.fig"

/*-----*/
PROMPT 1 1200, 4520, 5600, 220 ABORT KEY "choose list box option"
"Open BOB.DOC by double-clicking on the filename."

TAG"DZ2"
COUNT SET 1 2900, COUNT SET 2 2695, COUNT_SET 3 1200, COUNT_SET 4 220
GET MOUSE 0 MS DBL_CLICK 1 2 3 4
IF TRUE GOTO "DZ"

WAS MOUSE 0 MS CLICK 1 2 3 4
IF TRUE GOTO "DZ1"
WAS MOUSE 0 MS SHFT_CLICK 1 2 3 4
IF TRUE GOTO "DZ1"
WAS MOUSE 0 MS BUTTON_DOWN 1 2 3 4
IF TRUE GOTO "DZ1"
WAS MOUSE 0 MS BUTTON_UP 1 2 3 4
IF TRUE GOTO "DZ1"
WAS MOUSE 0 MS HOLD 1 2 3 4
IF TRUE GOTO "DZ1"

PROMPT 1 1500 4180 5000 440 ABORT KEY "double-click mouse"
"Position the mouse on BOB.DOC before
double-clicking."
GOTO "DZ2"

TAG"DZ1"
PROMPT 1 1500 4180 5000 880 ABORT KEY "double-click mouse"
"To double-click the mouse, briefly tap the mouse
button twice; do not pause between clicks and do
not hold the mouse button down. Try again to
double-click on BOB.DOC."

GOTO "DZ2"

TAG "DZ"
MESSAGE_OFF
PRESERVE_FILE 0 "BOB.DOC"

```



```
INVERT ON 0 2900 2695 1200 220
TAB KEY {BOB.DOC}
INVERT OFF
RETURN KEY
```

```
ON_QUIT_GOTO"Q3"
```

```
/*-----*/
PROMPT 1 1000, 3200, 6000, 1320 ABORT KEY "select Text"
"Let's change the greeting in this letter from \"Hi Robert\"
to \"Hi Bob\". To correct the greeting, select \"Robert\"
by pointing the mouse pointer to the \"R\". Press and hold
the mouse button down while moving the mouse pointer right
until \"Robert\" and only \"Robert\" is selected. When the
entire word is selected, release the mouse button."
```

```
HOLD_SEQ 300 660 100 220, 880 660 120 220, 0 220 6400 4400
1 "the start of \"Robert\"" "the end of \"Robert\""
```

```
/*-----*/
M PROMPT ORG 2000 3300
PROMPT I 1200, 3200, 5500, 220 ABORT KEY "type \"Bob\""
"Replace the selected text by typing \"Bob\"."
```

```
GET_TEXT 0 "Bob"
```

```
PAUSE_MODE 120
```

```
/*-----*/
PROMPT 0 1000, 1320, 6000, 3080 {c} "C=lesson menu"
"
Lesson Summary\r\n
\r\n
Congratulations, you have completed the mouse lesson.
You have learned in this lesson:\r\n
\r\n
- How to point the mouse pointer.\r\n
\r\n
- How to press a button by \"clicking\".\r\n
\r\n
- How to choose an option by \"double-clicking\".\r\n
\r\n
- How to select an item by \"dragging\" the mouse\r\n
pointer."
```

```
TAG "Q3"
MESSAGE OFF PICTURE_OFF
ESC KEY
IN MSGBOX "Save Changes"
IF FALSE_GOTO "Q2"
ALT_N
```

```
TAG "Q2"
RESTORE FILE 0 "BOB.DOC"
MOUSE OFF
GOTO"Q1" IN_FILE "MOUSE.DOC"
```


Script Command Reference

Command Index:

ALLOW_INHIBIT	suspend during unexpected events
CALL	call a subroutine
CHANGE_DIR	change the current directory
COUNT_ABOVE	test for counter above a value
COUNT_BELOW	test for counter below a value
COUNT_DEC	decrement the counter
COUNT_EQUAL	test for counter at a value
COUNT_INC	increment the counter
COUNT_SET	set the counter to a value
DELETE_DIR	delete a directory
DELETE_FILE	delete a file
DISK_SPACE	check for room on the disk
ESC_FLAG	allow the script to expect Escape
EXPECT_KEY	get a key from the user and check it
FILE_EXIST	check for a file's existence
GET_ARROWS	get arrows to the specified location
GET_DLGBOX_CMP	get keys to dialog box component
GET_KEY	get any key from the user
GET_LB_ITEM	get keys to list box item
GET_RB	get keys to the radio button
GET_TEXT	get the string from the user
GET_TO_MENU	get keys to the specified menu
GOTO	transfer control to the label
IF_FALSE_GOTO	transfer if the last check was FALSE
IF_TRUE_GOTO	transfer if the last check was TRUE
IGNORE_INHIBIT	do not suspend on unexpected events
IN_DLGBOX	is the dialog box active?
IN_FILE	specifies the file a TAG is in
IN_LISTBOX	is the list box active?
IN_MSGBOX	is the message box active?
INVERT_OFF	turn off any INVERTing region
INVERT_ON	invert a region of the screen
KEY_INTERVAL	pause a time period between events
KEY_WITHIN	check if the key is in a list
LOOP_TO	decrement count and loop if not zero
M_PROMPT_ORG	specify origin of automatic messages
MESSAGE_BUFFER	preallocate memory for screen saves
MESSAGE_OFF	restore the screen under the message
MESSAGE_ON	put a message on the screen
ON_QUIT_GOTO	designate transfer on quit
ON_TIMEOUT_CALL	designate transfer on timeout
OPTIONS	process a menu of options
PASS_KEY	pass a retrieved key to the application
PASS_WHILE	pass matching keys to the application
PAUSE_MODE	pause for a time period
PICTURE_OFF	restore the screen under the picture
PICTURE_ON	put a picture on the screen
POINT_TO	point to a spot with the hand icon
PRESERVE_DT_CFG	substitute DESKTOP.DFT for the .CFG

PRESERVE_FILE	subtitute a .DFT file for another
PROMPT	prompt the user for input
RESTORE_DT_CFG	restore DESKTOP.CFG
RESTORE_FILE	restore a preserved file
RETURN	return from a subroutine
RUN_RESOURCE	run a DeskMate resource
START_IN	start in a PDM other than DeskTop
TAG	mark routine
UNPACK_FILE	unpack a file

ALLOW_INHIBIT

ALLOW_INHIBIT is no longer supported.

CALL "label"

CALL is used to call a script subroutine.

Parameters

"label" is the name of the routine.

Special Notes

The script return stack only has room for eight (8) entries. Calls must not be nested more than eight deep. If the subroutine is not in the current file, the label must be followed by the proper **IN_FILE** designation.

Example

```
CALL "query_user",IN_FILE "user.evn"
```


CHANGE_DIR "directory"

CHANGE_DIR is used to force your tutorial or demo to execute from the designated directory.

Parameters

"directory" is the destination directory.

Special Notes

The directory must exist. The directory must either contain the entire pathname, or it must be a direct descendent of the current directory.

Example

```
CHANGE_DIR "test"
```


COUNT_ABOVE counter value

COUNT_ABOVE sets the TRUE condition if the count is greater than the specified value, FALSE otherwise.

Parameters

counter is the number of the counter from 0 through 9.
value is a decimal number.

Special Notes

The general purpose counter can be used to keep track of errors and for looping. See also, **COUNT_BELOW** and **COUNT_EQUAL**.

Example

```
COUNT_ABOVE 1 2  
IF_TRUE_GOTO "end"
```


COUNT_BELOW counter value

COUNT_BELOW sets the TRUE condition if the count is less than the specified value, FALSE otherwise.

Parameters

counter is the number of the counter from 0 through 9.
value is a decimal number.

Special Notes

The general purpose counter can be used to keep track of errors and for looping. See also **COUNT_ABOVE** and **COUNT_EQUAL**.

Example

```
COUNT_BELOW 1 2  
IF_TRUE_GOTO "end"
```


COUNT_DEC counter

COUNT_DEC decrements the counter value. Sets the TRUE condition if the result is not negative, FALSE otherwise.

Parameters

counter is the number of the counter from 0 through 9.

Special Notes

The general purpose counter can be used to keep track of errors.

Example

```
COUNT_DEC 9  
IF_TRUE_GOTO "end"
```


COUNT_EQUAL counter value

COUNT_EQUAL sets the TRUE condition if the count is equal to the specified value, FALSE otherwise.

Parameters

counter is the number of the counter from 0 through 9.
value is a decimal number.

Special Notes

The general purpose counter can be used to keep track of errors and for looping. See also **COUNT_ABOVE** and **COUNT_BELOW**.

Example

```
COUNT_EQUAL 1 2  
IF_TRUE_GOTO "end"
```


COUNT_INC counter

COUNT_INC increments the count value.

Parameters

counter is the number of the counter from 0 through 9.

Special Notes

The general purpose counter can be used to keep track of errors.

Example

```
COUNT_INC 4  
COUNT_EQUAL 4 3  
IF_TRUE_GOTO "end"
```


COUNT_SET counter value

COUNT_SET sets the count value.

Parameters

counter is the number of the counter from 0 through 9.
value is a decimal value.

Special Notes

The general purpose counters can be used to keep track of errors.

Example

```
COUNT_SET 9 3
TAG "loop"
  GET_KEY
  COUNT_DEC 9
  IF_TRUE_GOTO "loop"
```


DELETE_DIR "directory"

DELETE_DIR is used to delete directories which you may have created during your tutorial or demo.

Parameters

"directory" is the directory to be deleted.

Special Notes

The directory must either contain the entire pathname, or it must be a direct descendent of the current directory. The directory must of course be empty.

Example

```
DELETE_DIR "test"
```


DELETE_FILE DMCONFIG_flag "filename"

DELETE_FILE is used to delete a file.

Parameters

DMCONFIG_flag is 0 (zero) if the file to be deleted is in the current directory, 1 if the file is in the DMCONFIG directory.

"filename" is the pathname of the file to be deleted.

Special Notes

If "filename" does not contain path information, the file will be deleted within the current directory.

Example

```
DELETE_FILE 0 "test.doc"
```


DISK_SPACE number_of_bytes

DISK_SPACE is used to verify the indicated amount, **number_of_bytes**, of free space required by the tutorial or demo exists on the disk.

Parameters

number_of_bytes is the decimal number of bytes of the disk space required.

Special Notes

If **DISK_SPACE** is immediately followed by **IF_TRUE_GOTO** or **IF_FALSE_GOTO** playback execution will resume based on the returned state. If **DISK_SPACE** is not associated with a conditional **GOTO**, playback execution will terminate if the number of bytes indicated does not exist on the disk. The minimum number of bytes to check for is 1024 since modifying the directory structure can cause the directory information itself to grow by 512 bytes.

Example

```
DISK_SPACE 1024
```


ESC_FLAG Flag

ESC_FLAG allows the script to accept an escape key without quitting.

Parameters

Flag is set to one if **ESC_KEY** is expected, zero otherwise.

Special Notes

Be sure to set **ESC_FLAG** to zero when **ESC_KEY** is not expected, as there is no way to quit when **ESC_FLAG** is one.

Example

```
ESC_FLAG 1
EXPECT_KEY ESC_KEY
ESC_FLAG 0
```


EXPECT_KEY KEY

EXPECT_KEY will get keystrokes from the user until the correct key is input. If the key matches on the first try the **TRUE** condition will be set, if not **FALSE** will be set. If **FALSE** is set, any previously drawn message will have been removed.

Parameters

KEY is the value of the key expected.

Special Notes

EXPECT_KEY combines the functions of **GET_KEY** and **KEY_WITHIN** in a single call. If a specific key is expected, this call is more efficient.

EXPECT_KEY will prompt the user if the correct key is not typed. If the user types the wrong key twice, the keyboard is displayed with a finger pointing to the appropriate key.

Example

```
MESSAGE_ON 1 1000 1100 2000 440
"Press ENTER to continue"
EXPECT_KEY RETURN_KEY
MESSAGE_OFF
```


FILE_EXIST "filename"

FILE_EXIST is used to verify the existence of a file.

Parameters

"filename" is the pathname of the file to be located.

Special Notes

If "filename" does not contain path information, the file will be searched for within the current directory. If FILE_EXIST is immediately followed by IF_TRUE_GOTO or IF_FALSE_GOTO playback execution will resume based on the returned state. If FILE_EXIST is not associated with a conditional GOTO and the file does not exist, playback execution will terminate.

Example

```
FILE_EXIST "test.doc"
```


GET_ARROWS X Y X0 Y0 X1 Y1

GET_ARROWS will get arrows from the user until the location is reached. Keys which would cause the user to leave the bounding area will be accepted. Non-arrow keys will cause an error message to be displayed.

Parameters

X is the number of horizontal arrows, negative number for left arrows and positive for right arrows.

Y is the number of vertical arrows, negative numbers for up arrows and positive for down arrows.

X0 Y0 X1 Y1 designates the bounding area within which arrowing is allowed. The current cursor position should be thought of as (0, 0). **X0** and **Y0** should be less than or equal to zero. **X1** and **Y1** should be greater than or equal to zero.

Special Notes

You must specify the bounding area correctly or the counters will get confused. For instance, if the cursor is on the last line in Text, down arrows produce no action, so **Y1** must be zero.

Example

```
GET_ARROWS 3 4 -2 -1 4 5
```

Allows arrowing within 2 columns left and 1 line above and 4 columns right and 5 lines below of the point 3 columns right and 4 lines down.

GET_DLGBOX_CMP tab# "string"

GET_DLGBOX_CMP will pass keys from the user which move the focus to the specified dialog box component. If the component is a push button, this call will not return until the push button has been "pressed."

Parameters

tab# is the number of TABs which must be pressed to get to the component.

"string" is used to identify the component in any prompting messages caused by the user entering destructive keys.

Special Notes

The dialog box must be the selected component prior to this call. **GET_DLGBOX_CMP** will use the push button accelerator if the component is a push button. **UP_ARROWS** and **DOWN_ARROWS** will be accepted only when the current component is not a list box. This call should not be used if the dialog box has side scrolling list boxes.

Example

```
/* Display the Open dialog box */  
F2 {o} RETURN_KEY  
  
/* Go to the extension field */  
GET_DLGBOX_CMP 2 "Extension Editfield"
```


GET_KEY

GET_KEY will get a keystroke from the user.

Parameters

None.

Special Notes

GET_KEY is used when we do not know what to expect from the user. It is more efficient to use EXPECT_KEY if a preferred key is known. GET_KEY should generally be followed by KEY_WITHIN.

Example

```
GET_KEY  
KEY_WITHIN {a} .. {z} []  
IF_TRUE_GOTO "label"
```


GET_LB_ITEM "string"

GET_LB_ITEM will pass keys from the user which lead to selecting a list box item.

Parameters

"string" is the exact string contained within the list box. The length of the string must not be greater than 32 characters.

Special Notes

The list box must be the selected component prior to this call. GET_LB_ITEM will accept UP_ARROWS, DOWN_ARROWS, and character keys. This call cannot be used for side scrolling list boxes.

If the user types any improper keys, the following message will appear:

To go to the "string" option,
press the "appropriate arrow" key,
or press the "first letter of string" key
until the option is highlighted.

Example

GET_LB_ITEM "PR_DOC.DOC"

GET_RB button_number "string"

GET_RBT will accept arrow keys to move the focus to a particular radio button.

Parameters

button_number is the integer from 0 to nButtons-1 that specifies the button.

"string" is used by the automatic prompts when the user presses an unacceptable character. The string length must not be greater than 32 characters.

Special Notes

Control returns when the focus is on the desired radio button. If the user types something other than a valid key, a message stating "Please use only [appropriate] keys to get to the [string]" will be displayed, where [appropriate] lists the proper arrow keys and [string] is the user defined string.

Example

```
M_PROMPT_ORG 0 4000
F2 {r} RETURN_KEY {draw} RETURN_KEY
F6 {p} RETURN_KEY
GET_RB 14 "Brick Pattern"
```


GET_TEXT flag "string"

GET_TEXT will get a string from the user.

Parameters

flag is the integer 0 if you wish to be case insensitive, 1 if you wish to be case sensitive, and 2 if you only wish to accept 0 through 9.

"string" may be up to 63 printable characters.

Special Notes

All printable characters will be passed to the application until the length of the string is reached. The input will then be compared to the desired string, and a message asking the user to try again will be displayed if the result was not obtained. The routine will not end until the correct string is entered. RIGHT_ARROW, LEFT_ARROW, BKSPACE_KEY, and DELETE_KEY are accepted to aid the user in editing. Note that the cursor position is not known when the routine returns. If the user mistypes the phrase, the user will be prompted to correct the mistakes. If the user continues to make mistakes, or pauses too long between keystrokes, the user can press the space bar to have the string auto-typed.

Example

```
GET_TEXT 1 "Zimbabwe is a nice place to live."
```


GET_TO_MENU function_key item_number "menu string" "item string"
accelerator_key

GET_TO_MENU will get keystrokes from the user until the desired menu item is reached.

Parameters

function_key is the number of the function key which pulls down the menu.

item_number is the number of the item on the menu.

"menu string" is the string which will be used in all messages guiding the user to the menu.

The string must not exceed 32 characters.

"item string" is the string which will be used in all messages guiding the user to the item. The string must not exceed 32 characters.

accelerator_key is the menu item accelerator, use SPACE_KEY if none exists.

Special Notes

Messages will be displayed to guide the user to the correct menu item if any keys are detected which do not lead to the correct destination.

Example

```
GET_TO_MENU 4 4 "Text" "Center" CTRL_C
```

If the user enters keys which other than those which lead to the proper menu option, the following messages will appear:

To choose Center from the Text Menu,
first find the word Text on the menu bar across the top of the
screen. Following it is the function key which pulls down the Text
menu. Press that function key now.

The next step to choose Center from the Text Menu, is to highlight
it. Press down arrow until the Center option is highlighted.

To choose the Center option,
press ENTER.

GOTO "label"

GOTO forces script event execution to resume after the corresponding label.

Parameters

"label" is the string associated with the TAG where event execution is to resume.

Special Notes

If associated TAG statement does not exist within the same event file, the IN_FILE option should be used. If the TAG cannot be located, execution will terminate.

Example

```
GOTO "Run_Text"
```

```
F2 {r} RETURN_KEY  
{draw} RETURN_KEY
```

```
TAG "Run_Text"  
  F2 {r} RETURN_KEY  
  {text} RETURN_KEY
```


IF_FALSE_GOTO "label"

IF_FALSE_GOTO forces script event execution to resume after the corresponding label if a previous command has resulted in the **FALSE** condition.

Parameters

"label" is the string associated with the **TAG** where event execution is to resume.

Special Notes

If associated **TAG** statement does not exist within the same event file, the **IN_FILE** option should be used. If the **TAG** cannot be located, execution will terminate.

Example

```
FILE_EXIST "test.doc"
```

```
IF_FALSE_GOTO "Run_Text"
```

```
F2 {r} RETURN_KEY
```

```
{draw} RETURN_KEY
```

```
TAG "Run_Text"
```

```
    F2 {r} RETURN_KEY
```

```
    {text} RETURN_KEY
```


IF_TRUE_GOTO "label"

IF_TRUE_GOTO forces script event execution to resume after the corresponding label if a previous command has resulted in the **TRUE** condition.

Parameters

"label" is the string associated with the **TAG** where event execution is to resume.

Special Notes

If associated **TAG** statement does not exist within the same event file, the **IN_FILE** option should be used. If the **TAG** cannot be located, execution will terminate.

Example

```
FILE_EXIST "test.doc"  
IF_TRUE_GOTO "Run_Text"
```

```
GOTO "doesn't exist", IN_FILE "test2.evn"
```

```
TAG "Run_Text"  
  F2 {r} RETURN_KEY  
  {text} RETURN_KEY
```


IGNORE_INHIBIT

IGNORE_INHIBIT is no longer supported

MESSAGE_ON ScreenSave Xorg Yorg Text Yext **"Text of message"**

MESSAGE_ON displays an information box on the screen.

Parameters

ScreenSave is 0 if you do not want to save the screen background, 1 otherwise.
Xorg and Yorg specify the world coordinate origin of the information box.
Text and Yext specify the world coordinate dimensions of the box.
"Text of message" is the string to be displayed in the box.

Special Notes

MESSAGE_ON is used to communicate with the viewer. The message will remain on the screen until a MESSAGE_OFF or another MESSAGE_ON command is encountered.

The screen background of the message box will be preserved. If the size of the box exceeds the available memory for storing the background, the box will not be drawn and the FALSE condition will be set.

To place a message box on the screen during recording, use the "Shift-F1 m" command. A default size INFO_BOX will appear on the screen. The arrow keys are used to reposition the box. The CTRL+ARROW keys are used to resize the box. To edit the message press [TAB], to cancel it press [ESC], to accept it without a message press [TAB] [TAB]. When editing the message NEVER have two carriage returns in succession. If this is necessary use: [ENTER], SPACE, [ENTER] instead. To accept the message press [TAB], to cancel press [ESC]. This procedure will write the MESSAGE_ON, it's maprect, the message, a long pause (700 which is 7 seconds), and a MESSAGE_OFF command.

If you wish to use the arrow icons in your messages, use the string "\5\6" for a left arrow, "\6\7" for a right arrow, "\8\9" for an up arrow, and "\A\B" for a down arrow.

The accessory icon is "\T".

If you wish to specify a function key, you may use "\E\F" for F1, "\E\10" for F2, "\E\11" for F3, "\E\12" for F4, "\E\13" for F5, "\E\14" for F6, "\E\15" for F7, "\E\16" for F8, "\1B\1C" for F10.

Example

A message box starting at column 10, row 5 and is 40 columns wide and 2 rows tall would have the following map rectangle (characters are 100 world coordinates wide and 220 world coordinates tall).

```
MESSAGE_ON 0 1000 1100 4000 440
"This is a message box"
PAUSE_MODE 700,MESSAGE_OFF
```


ON_QUIT_GOTO "label"

ON_QUIT_GOTO forces script event execution to resume after the corresponding label when user presses Esc.

Parameters

"label" is the string associated with the **TAG** where event execution is to resume.

Special Notes

This instruction affects all following events until another **ON_QUIT_GOTO** is encountered.

Example

```
ON_QUIT_GOTO "End"
/* Any user quits after this point will goto the end */
F2 {r} RETURN_KEY
{draw} RETURN_KEY

TAG "End"
```


IN_DLGBOX "Title String"

IN_DLGBOX returns TRUE if the specified dialog box is active.

Parameters

"Title String" is the title displayed in the dialog box frame.

Special Notes

The "Title String" must match exactly.

Example

```
F2 {r} RETURN_KEY
```

```
IN_DLGBOX "Run File", IF_FALSE_GOTO "something wrong"
```


IN_FILE "filename.ext"

IN_FILE is associated with any label not within the current file.

Parameters

"filename.ext" is the name of the file containing the label.

Special Notes

IN_FILE should immediately follow a branching label.

Example

```
GOTO "label", IN_FILE "test.evn"  
IF_FALSE_GOTO "label", IN_FILE "test.evn"  
ON_QUIT_GOTO "label", IN_FILE "test.evn"  
IF_TRUE_GOTO "label", IN_FILE "test.evn"  
CALL "label", IN_FILE "test.evn"
```


IN_LISTBOX "Title String"

IN_LISTBOX returns TRUE if the specified list box is active.

Parameters

"Title String" is the title of the list box.

Special Notes

The "Title String" must match exactly.

Example

```
TAG "get to text list box"  
  IN_LISTBOX "TEXT", IF_TRUE_GOTO "on text list box"  
  EXPECT_KEY TAB_KEY  
  GOTO "get to text list box"  
TAG "on text list box"
```


IN_MSGBOX "Title String"

IN_MSGBOX returns TRUE if the specified message box is active.

Parameters

"Title String" is the title of the message box.

Special Notes

The "Title String" must match exactly.

Example

```
F2 {r} RETURN_KEY
```

```
IN_MSGBOX "Save Changes", IF_FALSE_GOTO "something wrong"
```


INVERT_OFF

INVERT_OFF is used to restore an inverted area of the screen.

Parameters

None.

Special Notes

INVERT_OFF will assure that the area is not left inverted if the area was flashing.

Example

```
INVERT_ON 1 1000 1000 4000 220  
PAUSE_MODE 500  
INVERT_OFF
```


INVERT_ON FlashFlag Xorg Yorg Xext Yext

INVERT_ON is used to invert an area of the screen.

Parameters

FlashFlag is 1 if you want the area to flash, 0 otherwise.

Xorg and **Yorg** are the world coordinate origins of the area to **INVERT**.

Xext and **Yext** are the extents of the area.

Special Notes

INVERT_ON will invert the specified area once a second.

Example

```
INVERT_ON 0 1000 1000 4000 220
```

```
PAUSE_MODE 500
```

```
INVERT_OFF
```


KEY_INTERVAL Interval

KEY_INTERVAL sets the time delay between the playback of each recorded event.

Parameters

Interval specifies the number of hundredths of seconds to pause between each playback event.

Special Notes

KEY_INTERVAL is used to pace the playback of recorded events. If the playback is terminated by the user pressing Esc, the pauses will be ignored.

Example

```
KEY_INTERVAL 50  /* Pause for 1/2 second between each event */
```


KEY_WITHIN token list

KEY_WITHIN will compare a keystroke from the user to the values in a token list. If the key is within the token list, the **TRUE** condition will be set, if not **FALSE** will be set.

Parameters

token list contains all of the values expected for the key. All tokens must be separated by spaces or commas, and the list must be terminated by the end of block token '['. The range token '...' may be contained within the token list.

Special Notes

KEY_WITHIN will operate on the last key retrieved. It should follow **GET_KEY** or **EXPECT_KEY**.

Example

```
GET_KEY
KEY_WITHIN {a} .. {z} SPACE_KEY []
/* returns true if a key from a to z is typed, or */
/* a space key is typed                               */
IF_FALSE_GOTO "label"

MESSAGE_ON 1 1000 1100 2000 440
    "You have hit an alpha key"
    GET_KEY RETURN_KEY
MESSAGE_OFF

TAG "label"
```


LOOP_TO counter "label"

LOOP_TO decrements the **COUNT** and forces script event execution to resume after the corresponding label if the **COUNT** is greater than zero.

Parameters

counter is the number of the counter from 0 through 9.

"label" is the string associated with the **TAG** where event execution is to resume.

Special Notes

The **COUNT** is used for looping, any instructions which affect it should be used inside the loop with great care. Loops can be nested.

If associated **TAG** statement does not exist within the same event file, the **IN_FILE** option should be used. If the **TAG** cannot be located, execution will terminate.

Example

```
COUNT_SET 2 3
TAG "loop 3 times"
GET_KEY PASS_KEY
LOOP_TO 2 "loop 3 times"
```


M_PROMPT_ORG Xorg Yorg

M_PROMPT_ORG sets the origin for all automatically displayed messages.

Parameters

Xorg and **Yorg** are the world coordinate origins of the messages.

Special Notes

This call should be made prior to **EXPECT_KEY**, **GET_ARROWS**, **GET_TO_MENU** or any other self prompting call.

Example

```
M_PROMPT_ORG 2000 4000
EXPECT_KEY {k}
```


MESSAGE_BUFFER Xtext Ytext

MESSAGE_BUFFER preallocates memory for subsequent **MESSAGE_ON** and **PICTURE_ON** calls.

Parameters

Xtext is the world coordinate width of the largest message or picture you plan on displaying.

Ytext is the world coordinate height of the largest message or picture you plan on displaying.

Special Notes

PICTURE_ON and **MESSAGE_ON** can both save screen backgrounds if memory is available to do so. **MESSAGE_BUFFER** allows you to "set aside" memory for this purpose if your application allocates all of available memory. If the size of the box exceeds the available memory for storing the background, the **FALSE** condition will be set.

Example

```
MESSAGE_BUFFER 2000 440
```


MESSAGE_OFF

MESSAGE_OFF removes the current message, and restores the screen background.

Parameters

None.

Special Notes

PICTURE_OFF and **MESSAGE_OFF** both restore screen backgrounds. Great care should be taken in the order of restoration when message boxes and pictures overlap.

The screen background of the message box will be preserved. If the size of the box exceeds the available memory for storing the background, the box will not be drawn.

Example

```
MESSAGE_ON 0 1000 1100 2000 440
```

```
"This is a message box"
```

```
PAUSE_MODE 700,MESSAGE_OFF
```


MESSAGE_ON ScreenSave Xorg Yorg Xext Yext

"Text of message"

MESSAGE_ON displays an information box on the screen.

Parameters

ScreenSave is 0 if you do not want to save the screen background, 1 otherwise.

Xorg and **Yorg** specify the world coordinate origin of the information box.

Xext and **Yext** specify the world coordinate dimensions of the box.

"Text of message" is the string to be displayed in the box.

Special Notes

MESSAGE_ON is used to communicate with the viewer . The message will remain on the screen until a **MESSAGE_OFF** or another **MESSAGE_ON** command is encountered.

The screen background of the message box will be preserved. If the size of the box exceeds the available memory for storing the background, the box will not be drawn and the **FALSE** condition will be set.

To place a message box on the screen during recording, use the "Shift-F1 m" command. A default size **INFO_BOX** will appear on the screen. The arrow keys are used to reposition the box. The **CTRL+ARROW** keys are used to resize the box. To edit the message press **[ENTER]**, to cancel it press **[ESC]**, to accept it without a message press **[TAB]**. When editing the message **NEVER** have two carriage returns in succession. If this is necessary use: **CR**, **SPACE**, **CR** instead. To accept the message press **[TAB]**, to cancel press **[ESC]**. The **MESSAGE_ON** will write the **MESSAGE_ON**, it's maprect, the message, a long pause (700 which is 7 seconds), and a **MESSAGE_OFF** command.

If you wish to use the arrow icons in your messages, use the string **"\5\6"** for a left arrow, **"\6\7"** for a right arrow, **"\8\9"** for an up arrow, and **"\A\B"** for a down arrow.

The accessory icon is **"\T"**.

If you wish to specify a function key, you may use **"\E\F"** for F1, **"\E\10"** for F2, **"\E\11"** for F3, **"\E\12"** for F4, **"\E\13"** for F5, **"\E\14"** for F6, **"\E\15"** for F7, **"\E\16"** for F8, **"\1B\1C"** for F10.

Example

A message box starting at column 10, row 5 and is 40 columns wide and 2 rows tall would have the following map rectangle (characters are 100 world coordinates wide and 220 world coordinates tall).

```
MESSAGE_ON 0 1000 1100 4000 440
"This is a message box"
PAUSE_MODE 700,MESSAGE_OFF
```


ON_QUIT_GOTO "label"

ON_QUIT_GOTO forces script event execution to resume after the corresponding label when user presses Esc.

Parameters

"label" is the string associated with the **TAG** where event execution is to resume.

Special Notes

This instruction affects all following events until another **ON_QUIT_GOTO** is encountered.

Example

```
ON_QUIT_GOTO "End"
/* Any user quits after this point will goto the end */
F2 {r} RETURN_KEY
{draw} RETURN_KEY

TAG "End"
```


ON_TIMEOUT_CALL seconds "label"

ON_TIMEOUT_CALL forces script event execution to call the corresponding label when lack of user input results in the **TIMEOUT** condition.

Parameters

seconds in the time period to wait for a key.

"label" is the string associated with the **TAG** where event execution is to resume.

Special Notes

To disable the timeout function, set the "seconds" length to zero, and supply any string as the label. This instruction affects all following events until another **ON_TIMEOUT_CALL** is encountered.

Example

```
GOTO "Start"
TAG "Prompt"
    MESSAGE_ON 0 1000 1000 1000 1000
    " Are you there?"
    GET_KEY
    MESSAGE_OFF
RETURN

TAG "Start"
    ON_TIMEOUT_CALL 5 "Prompt"
    /* Any user timeout after this point will call */
    /* the prompt routine                               */
    GET_KEY
    /* If the user doesn't enter a key in 5 seconds, */
    /* Prompt will be executed.                       */
```


OPTIONS ScreenSave number max_length Xorg Yorg

"Title String"

"Option1" "Tag1"

:

"OptionN" "TagN"

OPTIONS displays a menu of options, and processes it.

Parameters

ScreenSave is 0 if you do not want to save the screen background, 1 otherwise.

number is the number of options to be presented.

max_length is the length of the longest option.

Xorg and **Yorg** specify the origin to display the menu.

"Title String" is displayed on the first line of the menu.

"OptionN" is the string to be displayed for the option.

"TagN" is a **TAG** within the current event file to which control will transfer if the option is selected.

Special Notes

A string giving selection instructions will be displayed at the bottom of the box. Any previously displayed messages will be removed by this command.

Example

```
OPTIONS 1 3 20 1000 2000
```

```
"DeskMate Lessons"
```

```
"one"      "first"
```

```
"two"      "second"
```

```
"three"    "third"
```

```
TAG "first"
```

```
GOTO "end"
```

```
TAG "second"
```

```
GOTO "end"
```

```
TAG "third"
```

```
GOTO "end"
```

```
TAG "end"
```

```
MESSAGE_OFF
```


PASS_KEY

PASS_KEY will pass a "key" to the user.

Parameters

None.

Special Notes

PASS_KEY should generally follow **GET_KEY** or **EXPECT_KEY**. The key passed will be the last key value set by one of these instructions. If a key has not been previously retrieved, garbage may be passed to the application.

Example

GET_KEY, PASS_KEY

PASS_WHILE counter_number token_list

PASS_WHILE will compare keystrokes from the user to the values in a token list. Keys will be obtained from the user and passed on to the application as long as they appear in the token list, or the maximum is not exceeded. If the max is exceeded, the counter will contain zero. If the key is not within the token list, FALSE will be set. The **KEY** value will be set to the last key and processing of the Script file will continue.

Parameters

counter_number is the counter to be used.

token list contains all of the values expected for the key. All tokens must be separated by spaces or commas, and the list must be terminated by the end of block token '[]'. The range token '...' may be contained within the token list.

Special Notes

PASS_WHILE acts as if it were the following sequence:

```
TAG "start"  
  GET KEY KEY_WITHIN token list IF_FALSE_GOTO "continue"  
  PASS KEY  
  COUNT_DEC n COUNT_EQUAL n 0 IF_FALSE_GOTO "start"  
TAG "continue"
```

Example

```
COUNT_SET 1 10  
TAG "arrows"  
  PASS_WHILE 1 RIGHT_ARROW []  
  IF_TRUE_GOTO "enough keys?"  
  PROMPT 2000 2000 2000 660 {c} "Any Key" "You Pressed A Wrong Key"  
  PAUSE_MODE 100  
  MESSAGE_OFF
```

```
TAG "enough keys?"  
  COUNT_EQUAL 1 0 IF_TRUE_GOTO "good job"
```

```
TAG "prompt"  
  PROMPT 0 2000 2000 2600 440 RIGHT_ARROW "Right Arrow"  
  "Press #1 more right arrows"  
  IF_FALSE_GOTO "prompt"  
  MESSAGE_OFF  
  PASS_KEY  
  COUNT_DEC 1
```


COUNT_EQUAL 1 0
IF_TRUE_GOTO "good job"
GOTO "arrows"

PAUSE_MODE Duration

PAUSE_MODE pauses the playback for the indicated time period.

Parameters

Duration specifies the number of hundredths of seconds to pause the playback.

Special Notes

To pace the playback of recorded events, see **KEY_INTERVAL**. If the playback is terminated by the user pressing **Esc**, the pauses will be ignored.

Example

```
PAUSE MODE 700  
/* Pause for seven seconds */
```


PICTURE_OFF

PICTURE_OFF removes the current picture and restores the screen background.

Parameters

None.

Special Notes

PICTURE_OFF and **MESSAGE_OFF** both restore screen backgrounds. Great care should be taken in the order of restoration when message boxes and pictures overlap.

Example

```
PICTURE_ON 0 1000 1100 "picture.fig"  
PAUSE_MODE 700  
PICTURE_OFF
```


PICTURE_ON ScreenSave Xorg Yorg "picture.fig"

PICTURE_ON displays a graphics form on the screen.

Parameters

ScreenSave is 0 if you do not want to save the screen background, 1 otherwise.

XORG and **Yorg** specify the world coordinate origin of the picture.

"picture.fig" is the pathname of the Draw compatible file containing the Form to be displayed.

Special Notes

PICTURE_ON is used to communicate with the viewer. The picture will remain on the screen until a **PICTURE_OFF** or another **PICTURE_ON** command is encountered.

Example

```
PICTURE_ON 0 1000 1100 "picture.fig"
```

```
PAUSE_MODE 700
```

```
PICTURE_OFF
```


POINT_TO Xorg Yorg ScreenSave Direction

POINT_TO displays a hand icon with the index finger pointing to the designated coordinate.

Parameters

Xorg and **Yorg** specify the world coordinate origin of the picture.

ScreenSave determines if the background will be preserved, 1 for preserve, 0 for overwrite.

Direction is the direction the hand points, 0 is a left pointing hand, 1 is a right pointing hand, 2 is an up pointing hand, 3 is a down pointing hand.

Special Notes

If the screen background is saved, any previously displayed picture will be removed. The hand will remain on the screen until a **PICTURE_OFF** or another **PICTURE_ON** command is encountered.

If the size of the hand exceeds the available memory for storing the background, the hand will not be drawn and the **FALSE** condition will be set.

Example

```
POINT_TO 1000 1100 1 1
```

```
PAUSE_MODE 700
```

```
PICTURE_OFF
```


PRESERVE_DT_CFG

PRESERVE_DT_CFG makes an alias copy of the **DESKTOP .CFG** file in the **DMCONFIG** directory, and vectors all file i/o to the alias file.

Parameters

None.

Special Notes

PRESERVE_DT_CFG must be executed prior to the termination of **PLAY .PDM**.

Any combination of **START_IN**, **PRESERVE_FILE**, and **COPY_FILE** may precede it in the event file, but nothing else.

The file **DESKTOP .DFT** will be searched for in the current **.TUT** file, and copied to filename **DESKTOP .\$\$** in the **DMCONFIG** directory. If this can be done successfully, all file i/o calls to **DESKTOP .CFG** will be rerouted to **DESKTOP .\$\$**. If not, playback will terminate with an error. Re-routing will continue until the **RESTORE_DT_CFG** command is encountered, or playback terminates.

Example

PRESERVE_DT_CFG

PRESERVE_FILE DMCONFIG_flag "filename.ext"

PRESERVE_FILE copies the associated .DFT file in the .TUT file to filename.\$B\$ in the current directory. All file I/O calls to the original will be rerouted to the alias.

Parameters

DMCONFIG_flag is set to 1 if the file to be preserved is in the DMCONFIG directory. If the file to be preserved is in the current directory, set this to 0.

"filename.ext" is the name of the file in the DMCONFIG or current directory which you wish to preserve.

Special Notes

The indicated files .DFT counterpart will be searched for in the current .TUT file, and copied to filename.\$B\$ in the current directory if it exists. After this is done, all file I/O calls referencing the original will be re-routed to the alias. PLAY.PDM will terminate if the required .DFT file is not located in the .TUT file.

Example

```
PRESERVE_FILE 1 "PERSONAL.ADR"
```


PROMPT ScreenSave Xorg Yorg Xext Yext key_value
"key string"
"Text of message"

PROMPT displays an information box on the screen and pauses until the user enters a key.

Parameters

ScreenSave is 0 if you do not want to save the screen background, 1 otherwise.

Xorg and **Yorg** specify the world coordinate origin of the information box.

Xext and **Yext** specify the world coordinate dimensions of the box. An extra line will always be added to the box to display the quit and key strings.

key_value is the key expected from the user. If this is **ABORT_KEY**, no **EXPECT_KEY** will be performed.

"key string" is the string to be displayed in the bottom right hand corner of the box.

"Text of message" is the string to be displayed in the box.

Special Notes

PROMPT adds the ability to specify a variable value in messages. If the character '#' is encountered in the text string, followed by a number from 0 through 9, the contents of the corresponding counter will be substituted for the two characters.

If you wish to use the arrow icons in your messages, use the string "\5\6" for a left arrow, "\6\7" for a right arrow, "\8\9" for an up arrow, and "\A\B" for a down arrow.

The accessory icon is "\T".

If you wish to specify a function key, you may use "\E\F" for F1, "\E\10" for F2, "\E\11" for F3, "\E\12" for F4, "\E\13" for F5, "\E\14" for F6, "\E\15" for F7, "\E\16" for F8, "\1B\1C" for F10.

"\C4" can be used to turn Highlight on.

"\C1" can be used to turn Bold on.

"\C3" can be used to turn Underline on.

"\C0" turns any of the above attributes off.

"Esc=quit" will always be displayed in the bottom left hand corner of the box.

Example

```
PROMPT 1 1000 1000 2600 440 {c} "C=continue" "This is an example"
```


RESTORE_DT_CFG

RESTORE_DT_CFG deletes the alias file `DESKTOP.B` in the `DMCONFIG` directory, and restores `DESKTOP.CFG` as the primary file.

Parameters

None.

Special Notes

The `DESKTOP.B` file will be deleted from the `DMCONFIG` directory. After this is done, file i/o to `DESKTOP.CFG` will return to normal.

WARNING ... Although **RESTORE_DT_CFG** will close the temporary environment files associated with `DESKTOP.PDM`, `DESKTOP.PDM` will use the current screen configuration to write out `DESKTOP.CFG` when it exits. Therefore, playback must not exit while running Desktop, or the "real" configuration will be overwritten.

Example

`RESTORE_DT_CFG`

RESTORE_FILE DMCONFIG_flag "filename.ext"

RESTORE_FILE finds the alaised copy of the indicated file in the current directory, and deletes it. It then restores the original as the primary file.

Parameters

DMCONFIG_flag is set to 1 if the file to be restored is in the DMCONFIG directory. If the file to be restored is in the current directory, set this to 0.

"filename.ext" is the name of the file in the DMCONFIG directory which you wish to restore.

Special Notes

The alaised . \$B\$ file will be located in the current directory. After this is done, filename. \$B\$ will be deleted and **"filename.ext"** will be accesed on all subsequent file calls.

Example

```
RESTORE_FILE 1 "PERSONAL.ADR"
```


RETURN

RETURN is used to return from a Script subroutine.

Parameters

None.

Special Notes

The script return stack only has room for eight (8) entries. Calls must not be nested more than eight deep.

Example

RETURN

RUN_RESOURCE function_number "RES" "Parameter String"

RUN_RESOURCE executes the indicated function within the specified resource. The return value of the function will be stored in the **KEY** variable.

Parameters

function number is an integer specifying which of the resources routines to execute.

"RES" is the filename of the resource, you must not specify the extension.

"Parameter String" is an up to 64 character string which is passed to the resource.

Special Notes

The function number will be passed to the resource in the ax register. A short pointer to the parameter string will be at SS:BP. Playback will be suspended while the resource runs. The returned parameter from the resource will be placed into the global KEY variable.

Example

```
RUN_RESOURCE 0 "RES" "This is a string"  
KEY_WITHIN {q} [] IF_TRUE_GOTO "quit"
```


START_IN "program.pdm" "data.fil"

START_IN causes **PLAY.PDM** to call **dm_SetNextApp** to the specified program, rather than returning to **DeskTop**.

Parameters

"**program.pdm**" must be the name of the DeskMate application you wish to run.

"**data.fil**" should be the filename of the data file you wish to begin in. This should be set to " " if no file is desired.

Special Notes

START_IN must be executed prior to the termination of **PLAY.PDM**. Any combination of **PRESERVE_DT_CFG**, **PRESERVE_FILE**, and **COPY_FILE** may precede it in the event file, but nothing else.

Example

```
START_IN "TEXT.PDM" "MYFILE.DOC"
```


TAG "label"

TAG marks a point in the script where event execution may be transferred.

Parameters

"label" is the string associated with the **TAG** where event execution is to resume.

Special Notes

The "label" strings of **TAG** statements should be unique within the event file.

Example

```
FILE_EXIST "test.doc"  
IF_FALSE_GOTO "Run Text"  
F2 {r} RETURN_KEY {draw} RETURN_KEY
```

```
TAG "Run_Text"  
F2 {r} RETURN_KEY {text} RETURN_KEY
```


UNPACK_FILE DMCONFIG_flag "filename1" "filename2"

UNPACK_FILE is used to unpack a file from the compressed TUT file to disk.

Parameters

DMCONFIG_flag is 0 if the file is to be copied to the current direstory, 1 if the file is to be copied to the DMCONFIG directory.

"filename1" is the pathname of the file to be copied.

"filename2" is the name to copy the file to.

Special Notes

If **"filename1"** or **"filename2"** do not contain path information, the files will be copied within the current directory. Path information will be used if it is supplied. If **"filename2"** already exists, it will be overwritten.

You should always use **FILE_EXIST** to insure that the source file exists, as well as **DISK_SPACE** to insure that sufficient space exists on the disk.

Example

```
UNPACK_FILE 0 "test.doc" "test.bak"
```


Keystroke Definitions

/* CTRL'ed A-Z */

CTRL_A ... CTRL_Z

/* ALT'ed A-Z */

ALT_A ... ALT_Z

/* Arrow keys */

UP_ARROW

DOWN_ARROW

LEFT_ARROW

RIGHT_ARROW

SHFT_UP_ARROW

SHFT_DOWN_ARROW

SHFT_LEFT_ARROW

SHFT_RIGHT_ARROW

CTRL_UP_ARROW

CTRL_DOWN_ARROW

CTRL_LEFT_ARROW

CTRL_RIGHT_ARROW

ALT_UP_ARROW

ALT_DOWN_ARROW

ALT_LEFT_ARROW)

ALT_RIGHT_ARROW

SHFT_CTRL_UP_ARROW

SHFT_CTRL_DOWN_ARROW

SHFT_CTRL_LEFT_ARROW

SHFT_CTRL_RIGHT_ARROW

/* Function keys */

F1 ... F10

SHFT_F1 ... SHFT_F10

CTRL_F1 ... CTRL_F10

ALT_F1 ... ALT_F10

/* HOME, END, PGUP and PGDN keys */

HOME_KEY

END_KEY

PGUP_KEY

PGDN_KEY

SHFT_HOME_KEY

SHFT_END_KEY

SHFT_PGUP_KEY

SHFT_PGDN_KEY)

CTRL_HOME_KEY

CTRL_END_KEY

CTRL_PGUP_KEY

CTRL_PGDN_KEY

SHFT_CTRL_HOME_KEY

SHFT_CTRL_END_KEY

SHFT_CTRL_PGUP_KEY

SHFT_CTRL_PGDN_KEY

/* INSERT and DELETE keys */

INSERT KEY
DELETE_KEY

SHFT INSERT KEY
SHFT_DELETE_KEY

CTRL INSERT KEY
CTRL_DELETE_KEY)

/* ENTER, ESC, SPACEBAR, BACKSPACE and TAB keys */
RETURN KEY
ESC KEY
SPACE KEY
BKSPACE KEY
TAB_KEY

SHFT RETURN KEY
SHFT_SPACE KEY
SHFT_BKSPACE KEY
SHFT_TAB_KEY

CTRL_SPACE_KEY

ALT SPACE KEY
ALT_BKSPACE KEY
ALT_TAB_KEY

/* Miscellaneous keys */
BRK KEY
PRINT KEY
ABORT_KEY
ALT_EQUAL

/* Big select keys */
BIG_SLCT UP_ARROW
BIG_SLCT_DOWN_ARROW
BIG_SLCT_LEFT_ARROW
BIG_SLCT_RIGHT_ARROW
BIG_SLCT_HOME_KEY
BIG_SLCT_END_KEY

Tutorial Player - PLAY.PDM and DMPLAY.RES

Use the DeskTop's File Run command to play back a tutorial or demo. From a runtime execution, you may chose to provide the user with a tutorial and demo menu option which will invoke the tutorial or demo. Run `PLAY.PDM` and enter the base file name used in the tutorial. If you do not specify a file name for the tutorial, a screen will appear presenting all available tutorial files.

Pictures used in tutorials or demos can be a maximum size of 8K bytes. Larger pictures will simply not display and an error condition is not returned.

Demo Launcher - DEMO.PDM

The source file `MODIFY.C` in the `TOOLS\DEMO` directory must be modified to create a customized `DEMO.PDM` demo launcher for your demo. The `LOADMSG.H` include file contains an array of characters which define a Form Manager draw list that displays a message while the demo is loading.

To create your customized `DEMO.PDM` file, do the following:

- 1) To customize the graphics form for your demo
 - a) run Draw and open the file `LOADMSG.FIG`, change the string as required, and copy the revised picture to the clipboard.
 - b) run Drawlist, at the prompt, type the name of the array, 'loadmsg' (this string must be spelled exactly as shown here). Press the OK pushbutton to accept the name. A message box will display size information. Press the OK pushbutton to remove the message box. Use 'Save as...' on the 'File' menu to save the file under the name, 'loadmsg.h'.
- 2) To name the demo to be launched, in `MODIFY.C` change the following string to the name of your demo:

```
char sIntroTut[] = "DEMO.TUT";
```

- 3) If you must change the size of the box containing the message, you may want to change the world coordinates at which it is displayed.

```
int x_origin = 19 * CHAR_XEXT / 2;  
int y_origin = 5 * CHAR_YEXT;
```

- 4) If your demo is so large that all of the files will not fit on one disk, by not putting the demo file, the user will be prompted for a second disk which must contain the file. Change the message box strings if your demo requires this message box.

```
char InsertDisk2TitleStr[] = "Ready For Disk 2";  
char InsertDisk2MsgStr[] = "Please insert disk 2 of the Your Name \\  
Demonstration.";
```


Event Recorder - RECORD.PDM and DMRECORD.RES

The recording application, RECORD.PDM, is used to record a sequence of keystrokes or events which are played back at a later time. The playback application, PLAY.PDM, will simply playback the keystrokes entered during the recording session. If PLAY is begun at a state other than where RECORD was begun, the events issued will not correspond to what was recorded.

Use the File Run command from the DeskMate DeskTop to initiate a recording session. Run RECORD.PDM and enter the base file name of the file which will hold your recorded events, the extension will default to "EVN". If you do not use the proper extension, RECORD will issue an error message and terminate. If you do not specify a file name, you will be prompted to enter a file name.

The screen will be redrawn to erase the Run dialog box, all keystrokes are recorded until the session is terminated with the ALT+F10 key combination.

Special commands are provided during recording to enhance a script. These commands are described in detail in the Script Command Reference of Part 6, Writing Tutorials and Demos.

Shift+F1 M	allows you to create a message box for display during playback. See the MESSAGE_ON command description for details.
Shift+F1 C	allows you to add a comment to the event file to aid in the editing of the event file at a later time.
Shift+F1 F	allows you to chain event files together. This is most useful for creating a sequence of demos, and also allows you to chain event files to themselves for continuously running of demos.

General Rules of Recording:

A recording session should always be issued from the DeskMate DeskTop to guarantee a stable start.

We recommend the environment variable DMCONFIG point to a separate demo directory to insure that the user's data files such as calendar information, address books, etc. are not lost.

The RECORD application requires that a copy of the DESKTOP.CFG configuration file named DESKTOP.DFT reside in the directory pointed to by the environment variable DMCONFIG. The DESKTOP.CFG file is saved and replaced by DESKTOP.DFT during recording and playback. This insures that the playback sequence always begins from the same DeskTop configuration as the recording and does not alter the user's working DeskTop configuration. Store Demos should use the default DeskMate DeskTop configuration, otherwise simply copy your DESKTOP.CFG to DESKTOP.DFT.

To avoid having a playback session fail because of different DeskTop configurations, run applications from the DeskTop menubar rather than arrowing or tabbing to application boxes on the screen to execute the application. You can never be sure that the player's screen layout will match the recorder's.

Script File Interpreter and Compiler - DMEI.EXE and DMEC.EXE

The utility programs, DMEI.EXE and DMEC.EXE, allow for the editing of script files which are compiled into event files. The new script or event file will have the same name as the event or document file, and will destroy any file which has the same name without asking for verification to overwrite. The maximum size for an **event file** is 10K bytes.

DMEI TUTORIAL will create a script file TUTORIAL.DOC from the event file
TUTORIAL.EVN.

DMEC TUTORIAL will create an event file TUTORIAL.EVN from the script file
TUTORIAL.DOC.

The script file can be modified to add new events, delete events, remove or add pauses or messages, or perform any of the recording commands. When you have completed your modifications, run DMEC.EXE on the document file to create a new event file with your changes.

By editing your event files, you can verify that necessary files exist, that there is sufficient disk space to execute, copy files, delete files, change directories, and delete directories. These commands are provided to insure that your demo will run properly, and to insure that it restores the state of the machine to what it was before you began.

Tutorial Compression Tools - DMPACK.EXE and DMUNPACK.RES

The Tutorial Compression Utility merges all of the files needed by a tutorial into one tutorial file. The only rule is that the first event file to be executed must have the same base name as the tutorial file. For instance, for the DMINTRO.TUT tutorial the first event file executed is DMINTRO.EVN. The order files are packed into a tutorial is not important. The syntax for the command is

```
DMPACK tutorial_file <list of files in tutorial>
```

where *tutorial_file* is the name of the resultant tutorial file. Subsequent calls to DMPACK with the same tutorial file name will add the files to the tutorial. For this reason, you should always delete the old tutorial file BEFORE creating a new one.

The list of files in the tutorial follows, each file is delimited by a space.

The decompression algorithm resides in the DMUNPACK.RES resource which should be distributed with your product.

Appendix A
DeskMate 3 File Formats

"Appendix"

Contents

Introduction	A-1
Address Book/Phone List	A-3
Calendar	A-5
Draw	A-9
Filer/Form Setup	A-11
Text	A-15
Worksheet	A-21

Introduction

This document contains the file formats for the basic DeskMate 3.0 applications. The Address Book application and Phone List accessory share the same file. The Filer and Form Setup applications also share data files. The Address Book, Calendar, and Filer applications all use the Database Resource for their file input/output. The Draw application using the Form Manager in the Core Services Resource. Refer to the DeskMate Technical Reference for more information.

Each file contains a 22-byte Page Setup information header.

Byte	Description
0	Application identifier (*see defines below)
1 -> 3	Default file extension
4 -> 21	PGSETUP data structure (** see definition below)

*Application Identifiers

FILER_FILE	3
OFFICE_TEXT_FILE	13
WORKSHEET_FILE	14
DRAW_FILE	16
CALENDAR_FILE	17
DRAW88_FILE	20

** Page Setup Data Structure Definition

```
struct margin_defn
{
    char    left;        /* left margin (in characters) */
    char    lwidth;      /* line width (in characters) */
    char    linepp;      /* total lines per page */
    char    plinepp;     /* printed lines per page */
};
typedef struct margin_defn MARGIN;

struct pgsetup_defn
{
    char    mode;        /* NOTEBOOK, LANDSCAPE, PORTRAIT */
    MARGIN  mNotebk;     /* notebook margin defines */
    MARGIN  mLandscp;    /* landscape margin defines */
    MARGIN  mPortrait;   /* portrait margin defines */
    char    bDspace;     /* double space boolean flag */
    char    bPgpause;    /* pause between pages flag */
    char    bScontrol;   /* send control sequences flag */
    char    bGraphic;    /* graphic mode boolean flag */
    char    bText;       /* text mode boolean flag */
};
typedef struct pgsetup_defn PGSETUP;
```


Address Book/Phone List

The data file (PERSONAL.ADR) contains three (3) tables. The CONFIG and NAMES table definitions are static (their column definitions do not change). The DATA table's definition is dynamic and controlled by the user.

CONFIG :

This table contains the user's (or users' in the Shared file) configuration information.

UserId	Unique network user id, up to 10 characters.
LastList	Name of list last accessed by the user, up to 20 characters.
ConfigInfo	User's address book configuratio information, up to 200 characters.

NAMES :

This table contains the internal column names and the external names, those the user sees when building a form letter, for the address book data file.

Internal	Internal column names for the first 15 columns of the DATA table, up to 20 characters.
Users	Names the user sees for the first 15 columns of the DATA table, up to 20 characters.

DATA :

This table contains the user's address and phone numbers. The first 15 columns are static. Columns 16 through 40 are used for the user-defined lists. The table is sorted by the fields LastName/FirstName. The Database Manager call GET_COLUMN_NAMES may be used to retrieve the names of all the columns in the table. The return value is the number of columns in the table. The number of columns minus 15 is the number of user-defined lists in the Address Book.

Title	5 character title field (Mr., Mrs., etc.). (10 in DeskMate 3.3)
FirstName	18 character first name field.
LastName	20 character last name field.
Address	38 character street address field. (43 in DeskMate 3.3)
City	20 character city field.
State	8 character state field.
Zip	10 character zip code field.
CompanyName	29 character company name field.
WorkPhone	20 character work phone number field.
HomePhone	20 character home phone number field.
DateOne	8 character julian date format (stored) field.
NoteOne	12 character notes field.
DateTwo	8 character julian date format (stored) field.
NoteTwo	12 character notes field.
Notes	40 character notes field.
First List	1 character (Y/N is entry assigned to list).
...	
Last List	1 character (Y/N is entry assigned to list).

Calendar

The data file (PERSONAL.CAL) contains one table for every calendar of information stored in the file up to a maximum of 20 calendars. The default file contains one calendar table named PERSONAL. All newly created calendar tables are given user defined names.

Each time CALENDAR.PDM is run, a check is made to see which table was open the last time the program was exited. The bText element of the PGSETUP structure (struct pgsetup_defn) stored with the file PERSONAL.CAL is used as an index to the calendar table to open. The index should be greater than or equal to 1 and less than or equal to the number of tables minus 1 (DBCOLS table should not be included in the number of calendar tables) and correlate to the order of the table names returned by the call db_mgr(GET_TABLE_NAMES,&pInfo). DBCOLS table is the first table name returned in the buffer with an index of 0 and thus an index of 1 would relate to the first valid calendar table name.

Each calendar table contains 7 columns.

Column name	Length	Type	
Date	3	C (character)	
DayOfWeek	1	C (character)	
Duration	1	C (character)	(NOT USED)
StartTime	1	C (character)	
EndTime	1	C (character)	
Protected	1	C (character)	
Description	120	K (international character)	

The table sort order is by Date/DayOfWeek/StartTime/EndTime/Description.

Each calendar table has a configuration record. The configuration record is stored with data in the following three columns:

Date
Protected
Description

The Date column of the configuration record has the following values: 0x20, 0x20, 0xFF

The Protected column of the configuration record has the following value: 0x30

The Description column of the configuration record has values from the following cal_config_data data structure:

```
struct cal_config_data
{
    char StartHour;           /*first hour displayed on weekly grid */
    char Screen;              /* screen last displayed */
    char NumChanges;          /* number of changes made to calendar */
    char Date[3];             /* date of last change */
    char ChangedByName[8];    /* user name who made changes */
    char Code[11];            /* code to identify password */
    char CalendarPassword[16]; /* password associated with calendar */
};
```

Because the database cannot store values less than 0x07, we add a constant of 0x61 to all values which may be out of valid range.

The first byte of the configuration record Description column (StartHour) will be the constant value 0x61 added to the starting hour displayed on the weekly screen graph. The default value in a new file is 8, thus 0x69 would be stored in the database if this value has not changed.

The second byte of the configuration record Description column (Screen) will be the constant value 0x61 added to a constant for the screen type as follows:

YEARLY_SCREEN	1
MONTHLY_SCREEN	2
WEEKLY_SCREEN	3
DAILY_SCREEN	4

Thus the second byte will be 0x62, 0x63, 0x64, or 0x65 depending on which screen was displayed when CALENDAR.PDM was last exited.

The third byte of the configuration record Description column (NumChanges) will be the constant 0x61. This value will not change and is not used.

The fourth, fifth, and sixth bytes of the configuration record Description column (Date[3]) will all be the constants 0x61. These values will not change and are not used.

The next eight bytes of the configuration record Description column (ChangedByName[8]) will all be the constants 0x61. These values will not change and are not used.

The next eleven bytes of the configuration record Description column (Code[11]) will be the hex values representing the string "@PaSsWoRd@=". These values will not change and are not used.

No data is written to the configuration record Description column for the CalendarPassword[16] element of the structure.

There are three other types of records that may be added to the file: (1) events, (2) reminders, and (3) annual occasions

A maximum of 20 events per day may be added to each calendar table. A maximum total of 70 reminders and annual occasions may be added to each calendar table. The Description column is limited to 120 characters for events, 60 characters for reminders, and 30 characters for annual occasions.

The event, reminder, and annual occasion records are stored with data in the following six (6) columns:

Date
DayOfWeek
StartTime
EndTime
Protected
Description

Event records:

The `Date` column of an event record has the constant values of 0x61 added to the values for year, month, and day. The value for year will be the actual year minus 1980 (thus 8 for 1988), while the value for month will be between 1 and 12, and the value for day will be between 1 and 31. The `Date` column will be stored as: 0x61 + year, 0x61 + month, 0x61 + day

The `DayOfWeek` column of an event record has the constant value of 0x61 added to a value between 1 and 7 (1 = Sun, 2 = Mon, etc.). The value is stored as: 0x61 + `DayOfWeek`

The `StartTime` and `EndTime` columns of an event have the following value if both the start time and end time were blank in the dialog box: 0x80

If only the end time was blank in the dialog box, the `EndTime` column is stored as: 0xF0

All valid times stored in `StartTime` and `EndTime` columns will be stored as follows (where bit 7 is the most significant bit):

- | | |
|---------------------|---|
| bit 7 | - value 1 |
| bits 6,5,4,3, and 2 | - value representing hours 0 through 24 |
| bits 1 and 0 | - value representing quarter hours (0 = 0 minutes,
1 = 15 minutes, 2 = 30 minutes, 3 = 45 minutes) |

The time 12:00 am (midnight) is stored as 0x80 if it is a start time and 0xE0 if it is an end time. All other times have only one value regardless of whether the time is a start time or end time. Examples follow:

12:15 am	0x81
01:00 am	0x84
11:45 pm	0xDF

The `Protected` column of an event record has the following value if there is not an alarm associated with the event: 0x31

The `Protected` column of an event record has the following value if there is an alarm associated with the event: 0x71

The `Description` column of an event record has values from a string which is limited in length to 120 characters.

Annual Occasion records:

The `Date` column of an annual occasion record has the constant values of 0x61 added to the values for year, month, and day. The value for year will be 0, while the value for month will be between 1 and 12, and the value for day will be between 1 and 31. The `Date` column will thus be stored as: 0x61, 0x61 + month, 0x61 + day

The `DayOfWeek` column of an annual occasion record has the constant value of 0x61 added to the value 8 which identifies the record to not be an event (which has values between 1 and 7). Thus the value is stored as: 0x69

The `StartTime` column of an annual occasion record has the value `0x80` and the `EndTime` column has the value `0xBC` if the annual occasion date is not February 29 of a leap year. If the date of the annual occasion is February 29 of a leap year, the value in both the `StartTime` and `EndTime` column will be `0x80` added to the value of the year (4 for 1984, 8 for 1988, etc.). Thus the stored values for `StartTime` and `EndTime` will be:

<code>0x80 (or 0x80 + year)</code>	<code>StartTime</code>
<code>0xBC (or 0x80 + year)</code>	<code>EndTime</code>

The `Protected` column of an annual occasion record has the following value: `0x31`

The `Description` column of an annual occasion record has values from a string which is limited in length to 30 characters.

Reminder records:

The `Date` column of a reminder record has the constant values of `0x61` added to 0 for year, month, and day and is thus stored as: `0x61`, `0x61`, `0x61`

The `DayOfWeek` column of a reminder record has the constant value of `0x61` added to the value 8 which identifies the record to not be an event. Thus the value is stored as: `0x69`

The `StartTime` and `EndTime` columns of a reminder record has the following value in each column: `0x80`

The `Protected` column of a reminder record has the following value: `0x31`

The `Description` column of a reminder record has values from a string which is limited in length to 60 characters.

Draw

A Draw data file consists of the Page Setup header followed by a FORM_HEADER which describes the graphics form which follows. Following the graphics form is the palette information for the picture. The GUF Resource High-level File I/O calls are used to access the file, see the File I/O Manager section of the DeskMate Technical Reference for details.

See the Introduction section of this document for information on the Page Setup header.

See the Form Manager section of the DeskMate Technical Reference for more information about graphics forms.

See the Video Manager section of the DeskMate Technical Reference for information on retrieving and setting the video palettes.

Byte	Description
0 -> 21	File header.
22 -> 39	FORM HEADER data structure.
40 -> 43	FORM_SIZE_BUF data structure.
44 -> eoform	Graphics Form information (*see below).
eoform+1 -> eoform+64	Palette information (**see below).

* How to determine the size of the form

```
eoform = FORM_SIZE_BUF.list + FORM_SIZE_BUF.strokes + 44;
```

** Palette information format

Byte	Description
0	Palette number (COLOR1).
1	Red value (0-255).
2	Green value (0-255).
3	Blue value (0-255).
...	
60	Palette number (COLOR16).
61	Red value (0-255).
62	Green value (0-255).
63	Blue value (0-255).

Filer/Form Setup

Each file contains either two (2) or three (3) tables, depending on whether the file contains graphics or not. The LAYOUTS and GRAPHICS table definitions are static. The DATA table's definition is dynamic and controlled by the user.

LAYOUTS :

This table contains the descriptions of the Record and Report forms. It is made up of thirteen (13) columns. It is sorted by START_ROW/START_COL.

ITEM#	Unique sequential number (0-999), order items were added to the table.
FIELDID	Unique sequential number (0-999), order the fields were added to the table. For summary fields, contains the field id of the field being summarized. For report markers, id of marker (Header (0), Body (1), Summary (2), Footer (3)).
TYPE	Type of item (see Table A)
FORMAT1	Format Information (see Table B)
FORMAT2	Format Information (see Table C)
FMT_CHARS	For fields and summaries, the format string used when editing the field information in Filer. For static text, the text string.
DESCRIP	For fields and summaries, the name of the user gave the field or summary.
START_COL	For all items except report markers, contains the starting column of the item on the layout (0-131). For report markers, unused.
START_ROW	For all items except report markers, contains the starting row of the item on the layout (0-22). For items in the report, starting row is relative to the section item appears in. For report markers, line of form actual marker is on.
NUM_COLS	For all items, the number of columns used by the item (1-131).
NUM_ROWS	For all items except report markers, the number of rows used by the item (1-22). For report markers, number of lines in the section.
COL_OFFSET	Not used.
ROW_OFFSET	For fields and report fields, number of decimal places to the right of the decimal. For all other items, unused.

DATA :

This table may contain up to 22 columns which are defined by the user. Each column in this table is represented by a field entry in the LAYOUTS table. The Database Manager call GET_COLUMN_NAMES may be used to retrieve the names of all the columns in the table. The return value is the number of columns in the table. The table's sort order is determined by the user, the name of the index is "DATAINDEX".

GRAPHICS :

This table contains the binary graphics data in an encrypted format since the Database may contain only ASCII data.

RECORD
BITS

Table A

101	Static text
102	Field
103	Not used
104	Summary Field
105	Date template
106	Page number template
107	Report Marker
108	Report static text
109	Report field

Table B

Static text	Text attribute
Field	Label location and outlining
Summary Field	Label location and outlining
Date	Not used
Page number	Not used
Report Marker	Nbr of blank lines at top of section
Report static text	Text attribute
Report field	Label location and outlining

Table C

Static text	Always 1 (for Body section)
Field	Type of field
Summary Field	Type of summary field
Date	Not used
Page number	Not used
Report Marker	Nbr of blank lines at bottom of section
Report static text	Section of report its in (0-3)
Report field	Type of field

Text attribute :

NORMAL
BOLD
UNDERLINE

Label Location and Outlining :

100	Label to the left of field
101	Label centered at top of field
102	No label appears
103	Outlined field, label to the left of field
104	Outlined field, label centered at top of field
105	Outlined field, no label appears

Field Types :

201	Single-line, left justified
202	Single-line, right justified
203	Numeric
204	Multi-line
205	Internal use
206	Numeric with fixed decimal point

Summary Field Types :

401	Summation
402	Average
403	Count

Filer uses the same format strings as an editfield in the component manager. See page 4-18.

Text

The Text application can create an ASCII file as well as its own non-ASCII data file. The GUF Resource High-level File I/O calls are used to access the file, see the File I/O Manager section of the DeskMate Technical Reference for details.

See the Introduction section of this document for information on the Page Setup header.

ASCII (and IBM extended) characters:

0D 0A	Carriage return line feed combination used to:
	1. terminates a line in program source code or batch files
	2. end a paragraph in documents
1A	EOF End of file marker
20 -> FF	Visible or printable characters

Note 1: 09 (Tab), 0C (Form Feed), and other values below 20 are not allowed in the file. When reading in a file which is not a "Text application" file (does not include a Text application header at beginning of file), any characters other than 0D and 0A in the range between 00 and 1F inclusive will be converted to a space.

Note 2: ASCII files use default page setup settings. A header is not stored at the beginning of ASCII files.

A Text application file becomes a non-ASCII file once any of the following actions have been taken upon a previously ASCII file:

- Merge in a non-ASCII Text application document.
- Change the Page Setup information from the default settings.
- Paste in text which contains underlined or bold characters.
- Paste in a Draw application picture.
- Make selected text bold.
- Make selected text underlined.
- Indent or center a paragraph.
- Create a header or footer.
- Insert a page number field which will automatically number pages.
- Insert a today's date field which will print the system date.
- Insert a database field which will allow printing form letters.

Note: The above actions result in control characters being stored in the file. The file can be restored to an ASCII file with the "To ASCII" menuitem (i.e all non-ASCII modifications to the file will be removed).

Non-ASCII files have the following format:

Byte	Description
0 -> 3	Four byte Text application identifier (0D 44 4F 43)
4 -> 21	18 (decimal) bytes of page setup information
22 -> EOF	All ASCII and control data other than page setup

HEADERS AND FOOTERS

After reading in a non-ASCII file, you must determine if the file has a header, a footer, or both. This is done by searching the file for the EOF marker (1A). If an EOF marker is found before reaching the last byte of the file (excluding 1A's representing control information within pictures and margin settings), there is a header or footer. If two EOF markers are found before the last byte in the file, there is both a header and a footer. Following is a dump of four very simple files with different combinations of headers and footers:

File with no header or footer:

```
0000 0D 44 4F 43 00 01 2B 2D 2D 05 64 2D 2D 05 46 42 *.DOC..+--.d--.FB*
0010 3C 00 00 00 00 01 54 68 69 73 20 69 73 20 74 68 *<.....This is th*
0020 65 20 64 6F 63 75 6D 65 6E 74 2E 1A 00 00 00 00 *e document.....*
```

At byte 16 hex, the document starts. This is the only case where there is no control information preceding the document text (note the two 0D's in the following three examples which precedes the document text). At byte 2B, the file ends with the EOF marker (1A).

File with a header, but no footer:

```
0000 0D 44 4F 43 00 01 2B 2D 2D 05 64 2D 2D 05 46 42 *.DOC..+--.d--.FB*
0010 3C 00 00 00 00 01 48 0D 0D 54 68 69 73 20 69 73 *<.....H..This is*
0020 20 74 68 65 20 68 65 61 64 65 72 2E 1A 0D 0D 54 * the header....T*
0030 68 69 73 20 69 73 20 74 68 65 20 64 6F 63 75 6D *his is the docum*
0040 65 6E 74 2E 1A 00 00 00 00 00 00 00 00 00 00 *ent.....*
```

At byte 16 hex, the control information for the header starts. The first byte will be either 'H' or 'h' followed by two 0D's. At byte 19 hex, the text of the header starts. At byte 2C, the header ends with an EOF marker. At byte 2D, two 0D's provide control information before the start of the document which is at byte 2F. The document ends at byte 44 with an EOF marker.

File with a footer, but no header:

```
0000 0D 44 4F 43 00 01 2B 2D 2D 05 64 2D 2D 05 46 42 *.DOC..+--.d--.FB*
0010 3C 00 00 00 00 01 46 0D 0D 54 68 69 73 20 69 73 *<.....F..This is*
0020 20 74 68 65 20 66 6F 6F 74 65 72 2E 1A 0D 0D 54 * the footer....T*
0030 68 69 73 20 69 73 20 74 68 65 20 64 6F 63 75 6D *his is the docum*
0040 65 6E 74 2E 1A 00 00 00 00 00 00 00 00 00 00 *ent.....*
```

At byte 16 hex, the control information for the footer starts. The first byte will be either 'F' or 'f' followed by two 0D's. At byte 19 hex, the text of the footer starts. At byte 2C, the footer ends with an EOF marker. At byte 2D, two 0D's provide control information before the start of the document which is at byte 2F. The document ends at byte 44 with an EOF marker.

File with a header and a footer:

```
0000 0D 44 4F 43 00 01 2B 2D 2D 05 64 2D 2D 05 46 42 *.DOC..+--.d--.FB*
0010 3C 00 00 00 00 01 48 0D 0D 54 68 69 73 20 69 73 *<.....H..This is*
0020 20 74 68 65 20 68 65 61 64 65 72 2E 1A 46 0D 0D * the header..F..*
0030 54 68 69 73 20 69 73 20 74 68 65 20 66 6F 6F 74 *This is the foot*
0040 65 72 2E 1A 0D 0D 54 68 69 73 20 69 73 20 74 68 *er....This is th*
0050 65 20 64 6F 63 75 6D 65 6E 74 2E 1A 00 00 00 00 *e document.....*
```

At byte 16 hex, the control information for the header starts. The first byte will be either 'H' or 'h' followed by two 0D's. At byte 19 hex, the text of the header starts. At byte 2C, the header ends with an EOF marker. At byte 2D, the control information for the footer starts. The first byte will

be either 'F' or 'f' followed by two 0D's. At byte 30, the text of the footer starts. At byte 43, the footer ends with an EOF marker. At byte 44, two 0D's provide control information before the start of the document which is at byte 46. The document ends at byte 5B with an EOF marker.

The following is a sample of code used to determine if a header and/or footer exists:

```

/*-----*/
unsigned int num_bytes_read; /* size of file excluding header, number */
/* of bytes in text buffer, returned from */
/* fil_menu_open() */
unsigned char *save_original_tbuf; /* ptr to beginning of the text buffer */
/*-----*/
unsigned char *get_EOF_pointer(ptr) /*called by check_for_header_and_footer()*/
unsigned char *ptr;
{
    register unsigned char *p;

    p = ptr;
    while ( TRUE )
    {
        switch ( *p )
        {
            case END_OF_FILE:
                return(p);
                break;

            case START_PICTURE:
                p = PICTURE_END_POINTER(p);
                break;

            case START_MARGIN:
                p = MARGIN_END_POINTER(p);
                break;

            default:
                break;
        }
        ++p;
    }
}
/*-----*/

/*-----*/
check_for_header_and_footer() /* called after opening a non-ASCII file */
{
    unsigned char *p;
    unsigned char *tbuf;

    /* initialize flags to assume there is no header or footer */
    header_exists = NO_HEADER;
    footer_exists = NO_FOOTER;

    /* initialize tbuf to point to the beginning of the text buffer*/
    tbuf = save_original_tbuf;

    /* find the first EOF marker in the file */
    p = get_EOF_pointer(tbuf);

    /*if the EOF marker is not the last byte read, we have a header or footer*/
    if ( (unsigned)(p - tbuf) + 1 < num_bytes_read )
    {
        if ( *tbuf == 'H' )
            header_exists = HEADER_ON_ALL_PAGES;
        else if ( *tbuf == 'h' )
            header_exists = HEADER_ON_ALL_PAGES_EXCEPT_1ST;
        else if ( *tbuf == 'F' )
            footer_exists = FOOTER_ON_ALL_PAGES;
        else if ( *tbuf == 'f' )
            footer_exists = FOOTER_ON_ALL_PAGES_EXCEPT_1ST;

        num bytes read -= (unsigned)(p - tbuf) + 1;
        tbuf = p + 1;
        p = get_EOF_pointer(tbuf); /* find the 2nd EOF marker in the file */

        /* if the EOF marker is not the last byte read, we have a footer */
        if ( (unsigned)(p - tbuf) + 1 < num_bytes_read )
        {
            if ( *tbuf == 'F' )

```



```

        footer_exists = FOOTER_ON_ALL_PAGES;
    else if ( *tbuf == 'f' )
        footer_exists = FOOTER_ON_ALL_PAGES_EXCEPT_1ST;
    }
}
/*-----*/

```

Following is a list if #defines used in the above code and in future examples for the Text application:

```

/*-----*/
/* Character attribute switches */
#define BOLD_ON 0x13
#define BOLD_OFF 0x12
#define UNDERLINE_ON 0x11
#define UNDERLINE_OFF 0x10
/*-----*/

/*-----*/
#define START_PICTURE 1
#define END_PICTURE 2
#define PICTURE_CLIP_LENGTH(p) (*(int *) (p + sizeof(char)))
#define PICTURE_COLUMN(p) (*(int *) (p + sizeof(char) +
                                     sizeof(int)))
#define PICTURE_WIDTH_WORLD_COORDS(p) (*(int *) (p + sizeof(char) + 12 *
                                     sizeof(int) + sizeof(FORM_HDR)) + 100)
/* 100 kludges for fat lines */
#define PICTURE_HEIGHT_WORLD_COORDS(p) (*(int *) (p + sizeof(char) + 14 *
                                     sizeof(int) + sizeof(FORM_HDR)) + 100)
/* 100 kludges for fat lines */
#define PICTURE_START_POINTER(p) (p - (*(int *) (p - sizeof(int))) - 3 *
                                     sizeof(int) - sizeof(char))
#define PICTURE_END_POINTER(p) (p + PICTURE_CLIP_LENGTH(p) +
                                     sizeof(char) + 3 * sizeof(int))
#define PICTURE_DRAW_POINTER(p) (p + sizeof(char) + 2 * sizeof(int))
#define PICTURE_HEIGHT_CHAR_BLOCKS(p) (PICTURE_HEIGHT_WORLD_COORDS(p) /
                                     CHAR_YEXT + 1)
#define PICTURE_WIDTH_CHAR_BLOCKS(p) (PICTURE_WIDTH_WORLD_COORDS(p) /
                                     CHAR_XEXT + 1)
/*-----*/
#define START_MARGIN 3
#define END_MARGIN 4
#define MARGIN_START_POINTER(p) (p - 4 * sizeof(char))
#define MARGIN_END_POINTER(p) (p + 4 * sizeof(char))
#define MARGIN_FIRST_LINE_INDENT(p) (*(p + 1 * sizeof(char)))
#define MARGIN_LEFT_INDENT(p) (*(p + 2 * sizeof(char)))
#define MARGIN_RIGHT_INDENT(p) (*(p + 3 * sizeof(char)))

struct paragraph_margin_defn
{
    unsigned char first_line_indent; /* # of chars from left margin */
    unsigned char left_indent; /* # of chars for all lines except 1st */
    unsigned char right_indent; /* # of chars from right margin */
};
typedef struct paragraph_margin_defn PARAGRAPH_MARGIN;
/*-----*/
/* START_FIELD and END_FIELD enclose Date, Page #, and Address Book fields */
#define START_FIELD 5
#define END_FIELD 6

/* these values are stored as 1st byte after START_FIELD - identify field type */
#define DATE_FIELD_TYPE 0
#define DATE_FIELD_TYPE_1 8
#define PAGE_NUMBER_FIELD 9
/*-----*/

```


CHARACTER ATTRIBUTES

The Text application supports boldfaced and underlined text. Character attribute control characters are embedded between the characters where the change in character attribute is to take place. All characters following a 13 (BOLD_ON) will be displayed/printed in boldface and all characters following a 12 (BOLD_OFF) will be displayed/printed in normal type. All characters following a 11 (UNDERLINE_ON) will be displayed/printed with an underline and all characters following a 10 (UNDERLINE_OFF) will be displayed/printed without an underline.

DRAW APPLICATION PICTURES

Draw application pictures pasted into a Text application file are surrounded by the control characters 01 (START_PICTURE) at the beginning of the picture and 02 (END_PICTURE) at the end of the picture. The 01 (START_PICTURE) must be preceded by a 0D 0A (carriage return line feed combination) or a 02 (END_PICTURE). This is because each picture is considered to be a paragraph and thus must be preceded by one of these two end-of-paragraph markers.

The first two bytes after the 01 (START_PICTURE) are used to store (as an integer) the length of the picture data as it was copied from the clipboard. Use PICTURE_CLIP_LENGTH(p), where p is the pointer pointing to the 01 (START_PICTURE), to access this value.

The third and fourth bytes after the 01 (START_PICTURE) are used to store (as an integer) the column position at which the picture is to be displayed. Use PICTURE_COLUMN(p), where p is the pointer pointing to the 01 (START_PICTURE), to access this value.

The fifth byte after the 01 (START_PICTURE) is the first byte of the actual Draw application picture data. Use PICTURE_DRAW_POINTER(p), where p is the pointer pointing to the 01 (START_PICTURE), to access this pointer. For example, to display the picture in the Text application, the following call is made:

```
vid_draw_form(PICTURE_DRAW_POINTER(p),
              (PICTURE_COLUMN(p) - first_column_displayed) * CHAR_XEXT,
              *row * CHAR_YEXT);
```

At the end of the actual Draw application picture data, there are another two bytes of data which are used to store the length of picture data as it was copied from the clipboard. These length bytes are followed by the 02 (END_PICTURE).

Because the data used to draw a picture will most likely contain data which could be confused with control codes used for other purposes, it is necessary to skip over the entire picture when searching for specific control characters (such as the 1A end of file marker). Use PICTURE_END_POINTER(p), where p is the pointer pointing to the 01 (START_PICTURE), to find the address of the corresponding 02 (END_PICTURE). If searching backwards through the file, use PICTURE_START_POINTER(p), where p is the pointer pointing to the 02 (END_PICTURE), to find the address of the corresponding 01 (START_PICTURE).

PARAGRAPH MARGINS

Paragraph margin information is stored between 03 (START_MARGIN) and 04 (END_MARGIN) control characters. Margin information is always stored before the first visible/printable character (including spaces) of the paragraph.

The first byte after the 03 (START_MARGIN) is used to store first line indentation. Use MARGIN_FIRST_LINE_INDENT(p), where p is the pointer pointing to the 03 (START_MARGIN), to access this value.

The second byte after the 03 (START_MARGIN) is used to store the left margin indentation of all lines other than the first line of the paragraph. Use MARGIN_LEFT_INDENT(p), where p is the pointer pointing to the 03 (START_MARGIN), to access this value.

The third byte after the 03 (START_MARGIN) is used to store right margin indentation. Use MARGIN_RIGHT_INDENT(p), where p is the pointer pointing to the 03 (START_MARGIN), to access this value.

Because the data used to control margin settings may likely contain data which could be confused with control codes used for other purposes, it is necessary to skip over the margin when searching for specific control characters (such as the 1A end of file marker). Use MARGIN_END_POINTER(p), where p is the pointer pointing to the 03 (START_MARGIN), to find the address of the corresponding 04 (END_MARGIN). If searching backwards through the file, use MARGIN_START_POINTER(p), where p is the pointer pointing to the 04 (END_MARGIN), to find the address of the corresponding 03 (START_MARGIN).

FIELDS

The Text application uses 05 (START_FIELD) and 06 (END_FIELD) to enclose control data that is to be used as a single piece of data. There are fifteen Address Book database fields, two date fields, and a page number field. Fields can be stored anywhere in the document, but data within a field is treated as a unit and may not be modified.

For Address Book fields, the 05 (START_FIELD) is always followed by an asterisk, which is followed by the name of one of the Address Book field, which is followed by another asterisk, which is followed by the 06 (END_FIELD). If the user prints form letters, actual data from the database will be substituted (at print time only) for the data between the 05 (START_FIELD) and 06 (END_FIELD) inclusive for each letter to be printed.

In the date fields, the 05 (START_FIELD) is followed by either 07 (DATE_FIELD_TYPE_0) or 08 (DATE_FIELD_TYPE_1) which is followed by a string which represents the form the date will print out as, which is followed by the 06 (END_FIELD). At print time the data between the 05 (START_FIELD) and 06 (END_FIELD) inclusive will be substituted with the system date in the form specified (MM-DD-YYYY or MMM DD, YYYY).

In the page number field, the 05 (START_FIELD) is followed by 09 (PAGE_NUMBER_FIELD), which is followed by the string "###", which is followed by the 06 (END_FIELD). At print time the data between the 05 (START_FIELD) and 06 (END_FIELD) inclusive will be substituted with the page number of the current page being printed.

Worksheet

A Worksheet data file consists of the Page Setup header followed by a description of the pad, an array of the five (5) list structures, the cell definitions, and the ascii string data. The GUF Resource High-level File I/O calls are used to access the file, see the File I/O Manager section of the DeskMate Technical Reference for details.

See the Introduction section of this document for information on the Page Setup header.

Byte	Description
0 -> 21	File header.
22 -> 121	Column Widths for columns 0-99 (4 to 77)
122	Extreme row of pad (0-99)
123	Extreme column of pad (0-99)
124 -> 125	Ignore
126 -> 175	Array of 5 TOPLIST structures*
176 -> eof cells	Cell information structures**
eof cells + 1 -> eof	String data

*TOPLIST data structure

```
struct
{
    char *start;      /* address of the first cell in the list */
    char *end;        /* address of the first unused byte after the
                      last entry in the list */
    int size;         /* the size of the cell in bytes */
    int status;
    int num;          /* the number of entries in the list */
};
```

The lists, and their respective cells, are in the following order :

- 1) labels
- 2) numbers
- 3) formulas
- 4) inputs
- 5) text blocks

Ignore the start and end elements, they are the actual addresses of the data in memory when the file was last saved; these addresses must be resolved at load time.

The number of entries in the list num will always include the extra "dummy" cell entry which terminates each list of cell definitions. Therefore an empty worksheet will have one (1) cell of each type located at R100,C100.

**Cell Data Structures.

```
struct
{
    char row;         /* Cell row */
    char col;         /* Cell column */
    int status;
    char *strPtr;     /* offset to string */
} STRING_CELL;
```



```

struct
{
    char row;          /* Cell row          */
    char col;          /* Cell column      */
    int status;
    char *strPtr;      /* offset to string */
    char rows;         /* Number of rows   */
    char cols;         /* Number of columns */
} TEXT_CELL;

```

The CELL structure is used for formula, input, and numeric cells.

```

struct
{
    char row;          /* Cell row          */
    char col;          /* Cell column      */
    int status;
    char *strPtr;      /* offset to string */
    double value;      /* value of the cell */
} CELL;

```

The string stored is the ascii representation of (1) the label the user entered in a text cell, (2) the text the user entered in a text block, (3) the number entered in a numeric cell, (4) the formula entered by the user in a formula cell, and (5) the name of an input cell.

The offset to the string is from the start of the TOPLIST array (byte 126).

The strings are all null-terminated and start immediately after the last cell structure. To determine the end of the cells, sum the size of each of the lists (number of cells in list times the size of the cell type in bytes).

The following is an example of a worksheet file with one cell of each data type.

```

2931:0100 0E 57 4B 53 00 05 1E 28-1E 05 46 28 1E 05 46 42 .WKS...{..F(..FB
2931:0110 38 00 00 00 00 01 0A 0A-0F 14 0A 0A 0A 0A 0A 0A 8.....
2931:0120 0A 0A 0A 0A 0A 0A 0A 0A-0A 0A 0A 0A 0A 0A 0A 0A .....
2931:0130 0A 0A 0A 0A 0A 0A 0A 0A-0A 0A 0A 0A 0A 0A 0A 0A .....
2931:0140 0A 0A 0A 0A 0A 0A 0A 0A-0A 0A 0A 0A 0A 0A 0A 0A .....
2931:0150 0A 0A 0A 0A 0A 0A 0A 0A-0A 0A 0A 0A 0A 0A 0A 0A .....
2931:0160 0A 0A 0A 0A 0A 0A 0A 0A-0A 0A 0A 0A 0A 0A 0A 0A .....
2931:0170 0A 0A 0A 0A 0A 0A 0A 0A-0A 0A 09 04 1D 02 00 67 .....g
2931:0180 0C 67 06 00 0A 00 02 00-0C 67 28 67 0E 00 02 00 .g.....g(g....
2931:0190 02 00 28 67 44 67 0E 00-02 00 02 00 44 67 60 67 ..(gDg,.....Dg`g
2931:01A0 0E 00 02 00 02 00 60 67-70 67 08 00 00 00 02 00 .....gpg.....
2931:01B0 01 01 0A 00 AF 01 64 64-75 72 00 00 02 02 02 00 ....//ddur.....
2931:01C0 A7 01 77 BE 9F 1A 2F DD-5E 40 64 64 75 6D 00 00 'w>.../j^@ddum..
2931:01D0 20 69 6E 20 74 68 65 20-04 04 D5 00 91 01 3C DF in the ..U...<
2931:01E0 4F 8D 97 EE 6B 40 64 64-65 74 00 00 29 2E 0D 0A O..nk@ddet...-
2931:01F0 46 69 72 73 03 03 92 00-9B 01 00 00 00 00 00 00 Firs.....
2931:0200 59 40 64 64 20 20 00 00-32 30 20 63 68 61 72 61 Y@dd ..20 chara
2931:0210 06 01 01 00 A2 00 03 03-64 64 74 20 00 00 64 72 ...."....ddt ..dr
2931:0220 54 68 69 73 20 69 73 20-62 6C 6F 63 6B 20 6F 66 This is block of
2931:0230 20 74 65 78 74 20 64 65-66 69 6E 65 64 20 69 6E text defined in
2931:0240 20 74 68 65 20 73 70 72-65 61 64 73 68 65 65 74 the spreadsheet
2931:0250 2E 20 20 54 68 65 20 0A-74 65 78 74 20 69 73 20 . The .text is
2931:0260 77 6F 72 64 2D 77 72 61-70 70 65 64 20 61 75 74 word-wrapped aut
2931:0270 6F 6D 61 74 69 63 61 6C-6C 79 20 62 79 20 74 68 omatically by th
2931:0280 65 20 65 64 69 74 66 69-65 6C 64 20 0A 63 6F 6D e editfield .com
2931:0290 70 6F 6E 65 6E 74 2E 01-01 01 01 01 01 01 01 01 ponent.....
2931:02A0 01 01 01 01 01 01 01 01-01 01 01 01 01 01 01 .....
2931:02B0 01 01 01 01 01 01 01 01-01 01 01 01 01 01 01 .....
2931:02C0 01 01 01 01 01 01 01 01-01 01 01 01 01 01 01 .....
2931:02D0 01 01 01 01 01 01 01 01-01 01 01 01 01 01 01 .....
2931:02E0 01 01 01 01 01 01 01 01-01 01 01 01 01 01 01 .....
2931:02F0 01 01 01 01 01 01 01 01-01 01 01 01 01 01 01 .....
2931:0300 01 01 01 01 01 01 01 01-01 01 01 01 02 00 72 .....r
2931:0310 32 63 32 2B 72 33 63 33-00 49 6E 70 75 74 20 46 2c2+r3c3.Input F
2931:0320 69 65 6C 64 00 31 32 33-2E 34 35 36 00 4C 61 62 ield.123.456.Lab
2931:0330 65 6C 00 C4 0C 8B 0E C6-0C 8B 16 C8 0C 8E 1E C2 el.D...F...H...B

```