eNetwork Personal Communications Version 4.3 for Windows® '95, Windows '98, and Windows NT®

CM Mouse Support User's Guide and Reference

eNetwork Personal Communications Version 4.3 for
Windows® '95, Windows '98, and Windows NT®

CM Mouse Support User's Guide and Reference

Note

Before using this information and the product it supports, be sure to read the general information under "Appendix D. Notices" on page 157.

Second Edition (January 1999)

This edition applies to Version 4.3 of IBM eNetwork Personal Communications and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Figures

Chapter 1. Introduction

CM Mouse is a program which provides intelligent and programmable mouse support for 3270 and 5250 emulation sessions. It allows users of host applications to point-and-click (instead of ″hunt-and-peck″) to perform host functions. Depending upon how scripts are written for a host session, it is possible for CM Mouse users to point-and-click to send a PF keystroke or a complex set of keystrokes (up to 4000 characters long) to the host.

CM Mouse can be used to divide screens into areas called hot spots, and different tasks and functions can be performed by pointing-and-clicking on the various hot spots. Hot spots are not fixed function or fixed placement, but change dynamically as you move through a host application to give true context sensitive function to each host application screen. All of this is easily controlled through an easy-to-use scripting language. Many samples scripts are provided for popular host applications such as IBM OfficeVision (VM, MVS, and AS/400).

CM Mouse popup menus give you instant access to lists of options and functions with a simple point-and-click. Popup menus can simplify command selection, choose options, and automate repetitive tasks. The CM Mouse popup menu editor [1] gives a WYSIWYG (What You See Is What You Get) view of a popup menu and allows you to quickly and easily customize the menus to your own purposes, or to create new menus.

The CM Mouse scripting language is powerful enough to automate complex host interactions, yet simple enough for even novice users. For the advanced user, integration with the OS/2 REXX language gives even more flexibility and programming power. REXX programs can be seamlessly imbedded directly into CMMouse scripts.

CM Mouse actually consists of a family of programs. The programs are functionally similar, and a user familiar with CM Mouse in one environment can easily move to another. Except as noted in “Chapter 16. Cross Platform Compatibility” on page 143, script files written in one environment can be used in another without modification.

The following table summarizes the supported environments and emulators.

Table 1. Supported Environments and Emulators

OS/2 Version 2 or higher	IBM Personal Communications for OS/2 Version 4.0 or higher
	IBM Communications Manager/2 Version 1.0 or higher
	Advantis Passport/2
MS Windows 3.1	IBM Personal Communications/3270 Version 3.0 or higher (Windows Mode)
	Advantis Passport/Windows
Microsoft Windows 95	IBM Personal Communications Version 4.1 or higher
	IBM Global Network/Windows 95
Microsoft Windows 98	IBM Personal Communications Version 4.3
	IBM Global Network/Windows 98
Microsoft Windows NT	IBM Personal Communications Version 4.1 or higher
	IBM Global Network/Windows NT

1. The menu editor is currently available only on OS/2 systems.

Throughout this document the following symbols are used to denote sections which are applicable to only a single environment:

OS/2 for OS/2 only.

WIN for Windows 3.1, Windows 95, Windows 98, and Windows NT only.

Chapter 2. Installing CM Mouse

There are no specific installation requirements beyond those of the software specified in the previous chapter of this book.

CM Mouse requires approximately 1 megabyte of disk space and a Microsoft-compatible mouse driver.

Installation Procedures

For all supported releases, CM Mouse can be easily installed from the Personal Communications Version 4.3 CD-ROM using the provided graphical interface.

To install CM Mouse, do the following:

1. Select **Run** from either the File pulldown or the Task bar of the CD-ROM interface.
2. Enter **E:\Setup** in your Command Prompt box (where E is your CD-ROM drive).

See the ***Personal Communications CD-ROM Guide to Installation*** manual for more details regarding installation.

Chapter 3. Starting CM Mouse

This chapter describes the procedures for starting CM Mouse, along with optional command line parameters which can be specified.

Startup Procedures

OS/2

Locate the CM Mouse icon in the CM Mouse folder and double-click on it with the left mouse button. When CM Mouse initialization is complete, the ‘A’ session is maximized and made the active window. (This can be overridden with the CM Mouse configuration dialog, see “Chapter 4. Configuring CM Mouse” on page 9).

If you want to start CM Mouse automatically when your system is started, place a shadow of the CM Mouse program icon in the OS/2 ″Startup″ folder.

The emulator program does not have to be running when CM Mouse is started. If the emulator is not available, then CM Mouse will display a dialog message and wait up to 5 minutes. If the emulator sessions are still not available after 5 minutes CM Mouse will stop waiting and you will need to select **Reset->Host Connections** from the CM Mouse control panel after the emulation sessions are started.

WIN

Locate the CM Mouse icon in your desktop group and double-click on it with the left mouse button. When CM Mouse initialization is complete the ‘A’ session is maximized and made the active window. (This can be overridden with the CM Mouse configuration dialog, see “Chapter 4. Configuring CM Mouse” on page 9).

The emulator program does not have to be running when CM Mouse is started. If the emulator is not available, then CM Mouse will display a dialog message and wait up to 5 minutes. If the emulator sessions are still not available after 5 minutes CM Mouse will stop waiting and you will need to select **Reset->Host Connections** from the CM Mouse control panel after the emulation sessions are started.

Optional Command Line Parameters

In some operating environments CM Mouse is configured through the use of command-line parameters and/or environment variables. The command-line parameters allow the same configuration capabilities as the configuration dialog used in OS/2 and Windows. Parameters can be in upper, lower, or mixed case. Each parameter is explained below:

```
bdfdir=<dirpath>
bdf=<filename>
mmmdir=<dirpath>
editor=<pgmname>
initsess=<sessionid>
def=<fullpath><filename.ext>
quiet=yes|no
```

The equal sign (=) can be replaced with a minus sign (-) in any parameter specification. This can be useful when CM Mouse is invoked from a batch file which gives special meaning to "=".

bdfdir=<dirpath>
This parameter specifies a list of directories in which CM Mouse will look for button definition files. This parameter should include the full drive and path name of each directory, separated by semicolons (;). Directories may optionally include a trailing backslash. If not specified, this parameter defaults to C:\CMMOUSE\. All button definition files must appear in one of the directories listed.

Example:

```
bdfdir=d:\mydefs;f:\server\mousdefs
```

bdf=<filename>
This parameter specifies the file name of the primary CM Mouse button definition file (the first button definition file to be read). Only the file name should be specified (with no path name and no extension). The *bdfdir* parameter determines what directories will be searched, and in what order. The file extension BDF is always used and should not be specified. The name may optionally be prefixed with a *relative* path name which is appended to the *bdfdir* path (do not specify a leading backslash). In this case, the relative path is appended to each directory in the *bdfdir* path until the file is located.

This file is read when CM Mouse is started to determine what each mouse button is to do (see "Chapter 6. CM Mouse BDF Script Files" on page 27). If not specified, this parameter defaults to *CMMOUSE*. This file must exist in one of the *bdfdir* directories. (If all defaults are taken, the primary button definition file is CMMOUSE.BDF and it is read from the C:\CMMOUSE\ directory.)

mmmdir=<dirpath>
The *mmmdir* parameter specifies a list of directories in which CM Mouse will look for menu files (MMM = CM Mouse Menu). This parameter should include the full drive and path name of each directory separated by semicolons (;). Directories may optionally include a trailing backslash. If not specified, this parameter defaults to C:\CMMOUSE\. All menu files must appear in one of the directories listed. All menu files are assumed to have an extension of .MMM.

Example:

```
mmmdir=d:\mymenus;f:\server\menus
```

editor=<pgmname>
This parameter specifies the name of any text editor installed on your system. CM Mouse automatically invokes this editor to edit the BDF or MMM files whenever required. The editor is invoked with the file name to be edited as a command parameter (just as if it had been invoked at the command prompt). If this parameter is not specified, then it defaults to "e". The normal system search path is used to locate the editor, so this parameter does not need to specify the full path name of the program. [2]

Example:

```
editor=edlin
```

2. Note that this parameter (and all other parameters) will not perform properly if spaces are entered.

initsess=<sessionid>
This parameter specifies which host session (if any) should be maximized during CM Mouse initialization. The session ID specified should be a single letter ("a", "b"...). If this parameter is not specified, the default is "a". If you do not want any host session to be maximized, then specify "*****".

def=<fullpath><filename>
This parameter specifies the path and filename of a CM Mouse defaults file. The defaults file will be used to save customizing information such as colors and mouse speed settings. If not specified, this parameter defaults to C:\CMMOUSE\CMMOUSE.DEF. If specified, the full drive, path, and file name should be given. This file does not need to exist when CM Mouse is started; it will be created automatically when needed.

quiet=yes|no
The default for this parameter "no". If "yes" is specified then CM Mouse will not display a logo box or informational messages during initialization. If "no" is specified, CM Mouse will display various program and status information when it starts.

Examples

Example 1:

```
CMMOUSE bdf=main editor=edlin
```

This would start CM Mouse using the primary button definition file C:\CMMOUSE\MAIN.BDF. The EDLIN editor would be used for text editing.

Example 2:

```
CMMOUSE bdfdir=c:\cmmouse\buttons\ bdf=primary mmmdir=c:\cmmouse\menus\
```

In this example, the button definition files are read from the C:\CMMOUSE\BUTTONS\ directory, the primary button definition file is C:\CMMOUSE\BUTTONS\PRIMARY.BDF, and pop-up menu files are read from the directory C:\CMMOUSE\MENUS\. The default editor (e) would be used for text editing.

Example 3:

```
CMMOUSE bdfdir=c:\local\mymouse;f:\server\cmmouse mmmdir=c:\local\mymouse;
        f:\server\cmmouse
```

In this example two directories will be searched for BDF and MMM files. The first is on drive C in the \LOCAL\MYMOUSE directory and the second is on drive F in the \SERVER\CMMOUSE directory. This type of configuration is commonly used when a large number of BDF and MMM files are kept on a file server, but users on the network can keep a local version of any file they want to customize. By using this configuration, users can customize their files without affecting other users on the network and without replicating all the files they do not want to change. (See "Network Installation" on page 133.)

Environment (SET) Variables

Parameters can be specified to CM Mouse either by providing them when you start CM Mouse as described in "Optional Command Line Parameters" on page 5, or by executing SET statements before CM Mouse is started.

The environment variables for CM Mouse are:

```
CMMOUSE.BDF=<filename>
CMMOUSE.BDFDIR=<dirpath>
CMMOUSE.MMMDIR=<dirpath>
CMMOUSE.EDITOR=<pgmname>
CMMOUSE.INITSESS=<sessionid>
CMMOUSE.DEF=<fullpath>\<filename.ext>
CMMOUSE.QUIET=YES|NO
```

Each of these environment variables corresponds to one of the CM Mouse startup parameters described in "Optional Command Line Parameters" on page 5. If a particular parameter is specified both as an environment variable and on the command line when CM Mouse is started, the command line value is used.

For example, you could add the following lines to your AUTOEXEC.BAT file:

```
SET CMMOUSE.BDF=main
SET CMMOUSE.EDITOR=edlin
```

This would cause CM Mouse to use the primary button definition file C:\CMMOUSE\MAIN.BDF, and the EDLIN editor.

Chapter 4. Configuring CM Mouse

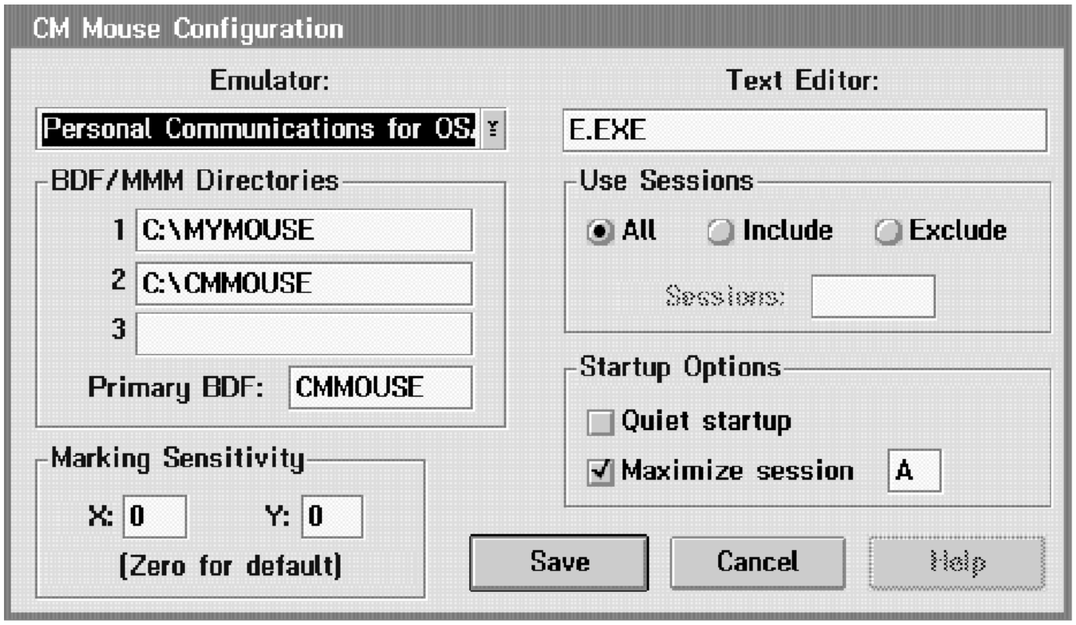
Note: You can configure CM Mouse by either following the steps below or by using the Optional Command-Line parameters in the previous chapter:

- select the **Configure** button on the CM Mouse installation dialog,
- run the CM Mouse Configuration Utility program from the desktop, or
- select the **Setup->Configuration** option from the CM Mouse Control Panel.

CM Mouse must be configured before it is run the first time. If CM Mouse has not been configured when it is started, it will automatically display the configuration dialog.

The CM Mouse configuration dialog tells CM Mouse some information it needs to properly startup and initialize. In particular you must tell CM Mouse where to find script files, which of the supported emulators you will be using, what host sessions you want to use with CM Mouse, and several other startup options.

The CM Mouse configuration dialog appears similar to the following:


CM Mouse Configuration

Emulator: Personal Communications for OS/

Text Editor: E.EXE

BDF/MMM Directories
1 C:\MYMOUSE
2 C:\CMMOUSE
3
Primary BDF: CMMOUSE

Use Sessions: [✓ All] Include Exclude
Sessions:

Startup Options
☐ Quiet startup
☑ Maximize session A

Marking Sensitivity
X: 0 Y: 0
(Zero for default)

Save Cancel Help

Figure 1. Configuration Dialog

Each item on the dialog is described below:

Emulator
: Select from the pulldown list, the emulator you will be using with CM Mouse. Only those emulators listed are supported by CM Mouse.

BDF/MMM Directories
: In the three numbered fields, enter the directories where CM Mouse is to find script and pop-up menu files (.BDF and .MMM). The directories will be searched in order from 1 to 3 when CM Mouse needs to locate a file. The *Primary BDF* field defines the name of the primary BDF script file (see "Chapter 6. CM Mouse BDF Script Files" on page 27).

Marking Sensitivity
: These values define how far you must drag the mouse with the left button

pressed before CM Mouse will recognize a clipboard marking operation. Some hardware devices (such as light pens which emulate a mouse) produce a large “skid” on the screen surface which appears as a left-button drag. Increasing these values can prevent such skids from being incorrectly interpreted as drag operations. The values are specified in screen pixels.

Text Editor
This field is used to specify what workstation text editor you want to use when editing script or pop-up menu files. Both the file name and extension should be specified. The editor will be invoked with the filename as a single parameter.

Use Sessions
These buttons allow you to restrict CM Mouse to specific host sessions. When the **All** button is selected (the default), CM Mouse will automatically connect to all available host sessions.

Selecting **Include** allows you to list in the **Sessions** field a specific list of sessions for CM Mouse to use. The list should be the short session IDs without any separation (for example, ″ACD″ would cause CM Mouse to use only sessions A, C, and D). If the sessions specified are not available when CM Mouse starts, it will wait for them to become available.

Selecting **Exclude** allows you to list in the **Sessions** field a specific list of sessions not to be used by CM Mouse. CM Mouse will use all the sessions available *except* those listed. The list should be the short session IDs without any separation (for example, ″AD″ would cause CM Mouse to use all sessions except A and D).

Startup Options
These options allow you to control what happens when CM Mouse is first started.

The *Quiet startup* option will cause CM Mouse to skip the usual startup logo dialog.

The *Maximize session* allows you to automatically maximize a particular host session when CM Mouse starts. Enter in the supplied field the single short session ID of the session to be maximized. If this option is not selected (unchecked), no host session will be maximized when CM Mouse starts.

Save Select this button when you have completed the configuration options. The options you specified will be saved. If you started the configuration dialog from a running copy of CM Mouse, the new values will be used immediately (if possible), or you will be notified that they will be used the next time CM Mouse is started.

Cancel
Select this button to cancel any changes to the CM Mouse configuration. The changes you made on this dialog will be discarded.

Chapter 5. Using CM Mouse

How you use CM Mouse depends a great deal on how you customize it and the scripts that you use. There are some general techniques and approaches that may prove helpful, however. Once CM Mouse has been started, you use the mouse pointer to point at some part of a host screen and press a button. Depending on how the scripts are written for that particular host screen, CM Mouse may do any of several things:

- Send a simple PF keystroke to the host (just as if you had pressed the PF key on your keyboard)
- Send a complex set of keystrokes up to 4000 characters long to the host (those keystrokes might include commands, PF keys, ENTER, or PA keys)
- Pop-up a menu on top of your host session from which you can select options or commands
- Do any combination of the preceding.

Although any mouse button can be configured to perform any host interaction, guidelines have been established to standardize the use of the mouse buttons. These standards ensure that the mouse buttons are configured in a consistent way for all host applications. For example, the RIGHT button is generally used to "backup" or "cancel" the current menu. See "Chapter 13. Button Usage Standards" on page 115 for additional information.

Writing CM Mouse Scripts

The following information lists the three steps you need to know in order to customize CM Mouse by modifying or writing CM Mouse scripts. "Chapter 14. Sample Button Definitions and Menus" on page 117 shows a step-by-step process for writing CM Mouse scripts using a real host screen as a sample.

Step 1

The first step is to make CM Mouse recognize the host screens in which you are interested. This is done in a CM Mouse *Button Definition File* (BDF). In a BDF you specify what it is about the host screen that makes it unique from all the other host screens you might use. This is usually done by keying on some small portion of text on the screen that does not change and that is unique to that screen. For example, most host applications display some sort of title line at the top of the screen. This can be used to make CM Mouse recognize that particular host screen and be able to distinguish it from all other host screens.

The syntax and format of BDFs are described in "Chapter 6. CM Mouse BDF Script Files" on page 27.

Step 2

The second step to writing a CM Mouse script is defining the keystrokes that are to be sent to the host when a button is pressed on a particular host screen. For example, you might want the RIGHT button to cause the VM FILELIST screen to scroll down and the LEFT button to scroll up. This part of the script is also specified in the BDF. (Note that the actions of the buttons should be defined according to the standards outlined in "Chapter 13. Button Usage Standards" on page 115.)

A BDF can do more complex things than defining screens and buttons. Host screens can be divided into *areas* (“hot-spots”) each of which has its own set of button definitions. For example, in the VM FILELIST screen, you might want the RIGHT and LEFT buttons to scroll as in the previous example, but if the mouse is pointing at a PF key description at the bottom of the screen, you might want that PF key sent to the host. This would allow you to “point-and-shoot” at PF keys on the bottom of the screen while maintaining the scrolling functions elsewhere on the screen.

Step 3

The third step to CM Mouse script writing is designing pop-up menus. Pop-up menus are invoked by specifying a special keyword in a button definition file. Using the FILELIST example described in Step 2, the LEFT button could be defined to display a pop-up menu rather than scroll the screen. To write a pop-up file you define a menu in a *CM Mouse Menu File* (MMM file). An MMM file consists of a title and lines of text. The text appears on the pop-up menu exactly as you specify it in the MMM file. With each line of text you specify what keys are to be sent to the host when that line of the pop-up is selected with the bounce bar cursor.

The syntax and format of pop-up menu files (MMMs) are described in “Chapter 7. CM Mouse Menu Files” on page 45. [3]

Writing CM Mouse scripts is made even easier with the large number of sample BDF and MMM files provided in the package. Sample scripts have been supplied for several popular VM applications including OfficeVision, RDRLIST, FILELIST, and others. The best way to learn how to write CM Mouse scripts is to try the samples, and then look at the files to see how they are done. More detailed information on the format of BDF and MMM files is given in Chapter 6. CM Mouse BDF Script Files and Chapter 7. CM Mouse Menu Files. A number of sample BDF and MMM files are explained in detail in Chapter 14. Sample Button Definitions and Menus.

Using Pop-up Menus

CM Mouse pop-up menus appear in a window on the screen similar to those shown in Figure 2 on page 13. There is a title area at the top, and any number of selection items in the main body of the menu. To make a selection from a pop-up menu, a *bounce bar* cursor will appear on the menu. The bounce bar is a highlighted item on the menu. By moving the mouse pointer up and down the menu, you can move the bounce bar to highlight any item on the menu. To select an item from a pop-up menu, position the bounce bar over that item and press the LEFT mouse button. To cancel a pop-up menu (dismiss the menu without selecting any items at all), position the mouse cursor anywhere in the menu and press the RIGHT button.

To move the pop-up, you can point to the title bar, press the LEFT button, and drag the window to a new location. (Pop-up windows can only be moved under OS/2 and Windows.; DOS pop-up menus cannot be moved.)

3. OS/2 users can create or modify popup menus using the CM Mouse Menu Editor without learning the syntax and format of .MMM menu files. See “The CM Mouse Menu Editor” on page 111.

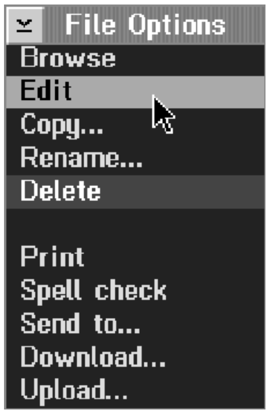


Figure 2. Sample Pop-up Menu

Canceling Pop-up Menus

While a pop-up menu is being shown, the pop-up window is the “active” window and the host window is disabled. You must either select an item on the pop-up menu (LEFT button) or cancel the pop-up (RIGHT button) to continue working in the host session. Of course you can switch to any other window in the usual way; only the host session is disabled. A pop-up menu can also be canceled by selecting the system-menu icon from the upper left corner of the pop-up and selecting the *Close* option from the pull-down menu.

Pop-up System Menus

CM Mouse pop-up menus contain a system-menu icon in the upper left corner. When this icon is selected, a pull-down menu appears with several options that are useful on pop-up menus (see Figure 3).

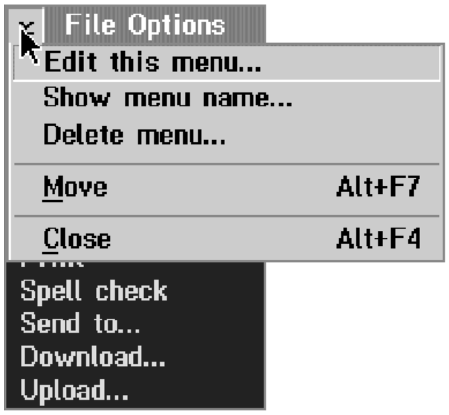


Figure 3. Pop-up System Menu

Following are the pop-up system menu options:

Edit this menu
When this option is selected, the file containing the current pop-up menu definition is copied to the first directory in the BDF/MMM path and the editor

is invoked on that file. [4] If there is only one directory in the path, or if the file already exists in the first directory, no copy is performed and the editor is simply invoked on the file. The particular text editor used is the editor specified in the CM Mouse configuration dialog.

The copy function of this option is useful when there is a large set of pop-up menu files and you want to change only a few of them. By using two directories in the BDF/MMM path (the first to keep only the files you change, the second containing the complete set), you can easily change only the pop-up menus you are interested in without modifying any of the original files. This is particularly useful when a set of pop-up menu files resides on a remote file server, and you want to customize some of them for your own use. If you were to edit the files on the file server, everyone using the server would be affected by your changes. By first copying the files to your own disk, you can make the changes you want without affecting anyone else and without having to copy all the files. (See “Network Installation” on page 133.)

A typical scenario would have a large set of pop-up menu files on a remote file server for all users to access. Each user’s BDF/MMM path would be set in the configuration dialog (“Chapter 4. Configuring CM Mouse” on page 9) to values like:

```
D:\MY\CMMOUSE
S:\PROD\CMMOUSE
```

The D:\MY\CMMOUSE directory is initially empty and is on a local fixed disk. The S:\PROD\CMMOUSE directory contains a complete set of MMM (pop-up menu) files and exists on a remote file server to which you might have read-only access. If you are satisfied with the pop-up menus provided and do nothing to change them, then the configuration will remain this way. However, if you wanted to change a particular pop-up menu, you would display the menu and pick the *Edit this menu* option. The file would be copied from the file server to your local disk, and the editor would be invoked. You could then change the pop-up menu and save it. The next time you display that menu, it is taken from your local disk and has the changes you made. If you edit the file again, the copy operation will *not* be done and the editor will simply be invoked on your local version of the file. In this way, you can continue to refine the pop-up menu by selecting the *Edit this menu* option.

Show menu name
When this option is selected, the actual file name from which the menu was retrieved is shown. This can be used to determine which of several directories in the BDF/MMM path list was used to find the file.

Delete menu
When this option is selected, the current pop-up menu file is deleted (after a confirmation dialog). This option can be used to delete a local copy of a pop-up menu after the *Edit this menu* option has been used.

Move When this option is selected, the pop-up menu can be moved. This is the same as pointing to the title bar, pressing a button, and dragging the window.

Close When this option is selected, the pop-up menu is canceled. This is identical to clicking the RIGHT mouse button.

4. OS/2 users may choose to use the CM Mouse Menu Editor or a text editor when editing menu files. To choose between the menu editor and a text editor, use the ″Setup->Options″ menu on the CM Mouse control panel.

Using the Keyboard on Pop-up Menus

OS/2

It is usually most convenient to use the mouse to select items on pop-up menus or to cancel them. However, you can use the keyboard for any functions which can be done with the mouse. The keyboard functions are:

ENTER or SPACE-BAR:
These keys are used to select the currently highlighted menu item. This is equivalent to pointing to the item with the mouse and pressing the LEFT button.

ESC, PF3 or PF12:
These keys cancel the pop-up menu. This is equivalent to using the RIGHT mouse button.

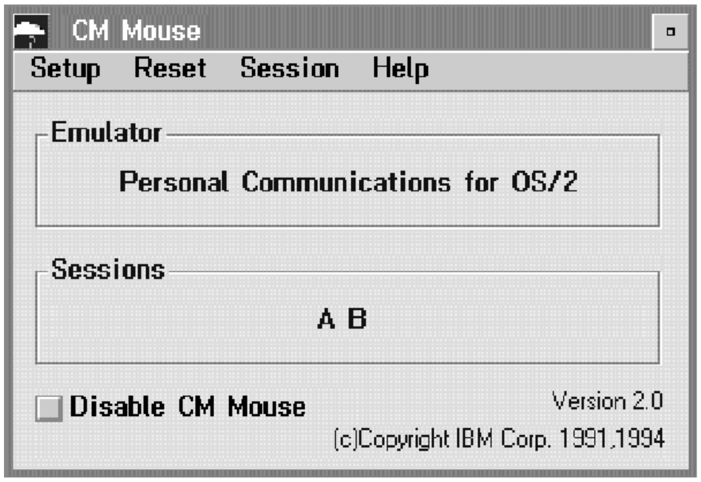
UP-ARROW:
The upward cursor movement key will move the bounce bar up one line on the menu.

DOWN-ARROW:
The downward cursor movement key will move the bounce bar down one line on the menu.

The CM Mouse Control Panel

CM Mouse has a “control panel” to control various aspects of operation such as the color of pop-up menus.

To display the control panel, double-click on the CM Mouse icon, or select the icon and pick the *Restore* option from the menu. A window similar to that appears in Figure 4.



Personal Communications for OS/2

A B

Version 2.0
(c)Copyright IBM Corp. 1991,1994

Figure 4. CM Mouse Control Panel

The control panel shows the current status of CM Mouse. The box labeled *Emulator* shows the emulator for which CM Mouse has been configured. The box labeled *Sessions* shows the emulator sessions to which CM Mouse is currently connected.

The *Disable CM Mouse* checkbox can be used to disable CM Mouse on all host sessions without closing the CM Mouse program. When CM Mouse is disabled, all mouse clicks on host sessions are handled by the native emulator; CM Mouse will not process any mouse clicks on any host sessions. This feature can be useful for special situations in which you need to use some mouse-related feature of the emulation program.

The following describes the function of each item on the menu bar at the top of the CM Mouse control panel.

Setup->Configuration
This option will display the CM Mouse Configuration menu (see "Chapter 4. Configuring CM Mouse" on page 9).

Setup->Options
This option will display the CM Mouse Option menu (see Figure 5). Each option may be toggled on or off by clicking on the checkbox. The options are:

- **Double-Click Support:** This option will allow double click mouse actions to be recognized on the host screen. Enabling this option will slow down response to single clicks.
- **Busy Mouse Pointer:** This check box enables the "busy pointer" that CM Mouse uses when it is busy processing a button click. When this feature is enabled, CM Mouse will change the pointer shape to a small mouse while it is processing a button click. This is a visual indicator to the user that CM Mouse is busy. When this feature is disabled, CM Mouse does not change the pointer shape when busy.

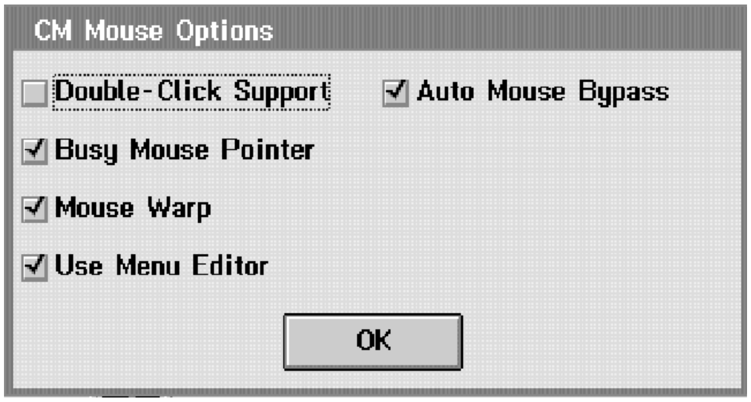


Figure 5. Options Menu

- **Mouse Warp:** This check box enables the "mouse warping" function of CM Mouse. When mouse warp is enabled (the box is checked), CM Mouse automatically positions the mouse pointer on the first line of pop-up menus. When a pop-up menu item is selected or the pop-up is canceled, the mouse pointer is returned to its original location. This effect of moving the mouse pointer is referred to as *mouse warping*. Some people like this function because it reduces the number of mouse movements required to select items on pop-up menus. Others prefer to disable this function and maintain a strict relationship between the position of the mouse on the table and the mouse pointer on the screen.

- **Use Menu Editor:** (OS/2 only) This check box enables the automatic use of the CM Mouse Menu Editor when selecting the **Edit this menu...** option of a popup menu. If this option is disabled, a text editor will be used to edit popup menus.
- **Auto Mouse Bypass:** When this option is enabled, CM Mouse will automatically pass mouse events to the emulator whenever the mouse cursor is anything other than the standard arrow. This feature allows certain emulator mouse functions to be used even when CM Mouse is active (for example, if the emulator changes the pointer to a cross-hair during GDDM graphics functions then CM Mouse can automatically pass mouse events to the emulator to be processed as GDDM pointer operations).

  When this option is disabled CM Mouse will always capture mouse events on the host emulator window (and thus these events will not be processed in the usual way by the emulator).

Setup->Popup Menu Colors

When this action is selected, the pop-up colors panel is displayed (Figure 6).

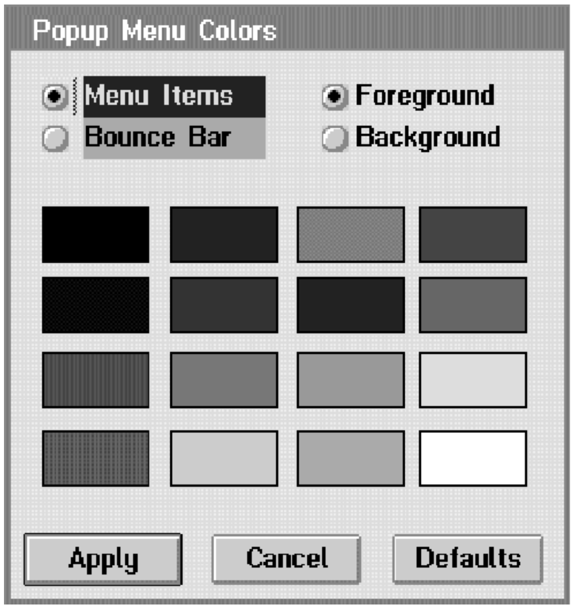


Figure 6. Pop-up Menu Colors Panel

The colors set on this panel will be used for pop-up menus that do not specify a specific color scheme. These will be the default pop-up menu colors. Colors can be set for four different parts of pop-up menus (foreground and background colors for each of the bounce bar and menu item fields).

To set a specific color, select the appropriate buttons at the top of the menu, and then select one of the color buttons. For example, to change the bounce-bar background color to red, select the **Bounce Bar** and **Background** buttons at the top of the menu, and then select the red-colored button. A sample of the menu colors is shown so that you can immediately see the effects of your color choices.

To use the color scheme you have set up, select the **Apply** button. The next time you pop up a menu that does not have a color scheme specified in the menu file, the new colors will be used. To cancel the changes you made, select the **Cancel** button. To set up the default CM Mouse colors, select the **Default** button. Your color choices will be saved when you stop CM Mouse.

Reset->Scripts
When this option is selected, all current screen and button definitions (scripts) are discarded and the primary BDF script file is reread. If the primary BDF file contains any INCLUDE statements, the included BDF files are also read. Since CM Mouse normally only reads scripts during initialization, this option must be used whenever a BDF file is modified.

Reset->Host connections
When this option is selected, CM Mouse will briefly connect to each host session, thereby rediscovering any new sessions started since CM Mouse was initialized. The list of active sessions shown on the control panel will be updated. This option must be selected whenever you start a new host session.

Session->Screen Map
OS/2 This option will start the CM Mouse Screen Map feature (see "CM Mouse Screen Map Facility" on page 20).

Session->Info
OS/2 This option will show various technical information about a specific host session. This information is generally used for debugging and problem reporting.

Help->Online book
OS/2 This option will display the CM Mouse User' Guide.

Help->About
This option will display version and copyright notices.

The Host System Menu

CM Mouse supports a host system menu. CM Mouse automatically intercepts mouse clicks on the system icon of the host window (the small square in the upper left corner of the window). Instead of pulling down the standard emulator window function list, a special CM Mouse pop-up menu is invoked instead (see Figure 7 on page 19).

You can customize this menu in the same way as any other CM Mouse pop-up menu. A default menu is provided in the package and contains the most commonly used emulator functions (CMMOUSE.MMM). Note that this menu is completely independent of the current host application; the same menu is always invoked when the system icon is selected. This allows you to place commonly used "utility" functions on the menu and have them always available no matter what host application is being used. For example, the emulator clipboard functions are provided as well as several other commonly used functions. You can modify this menu to suit your own needs.

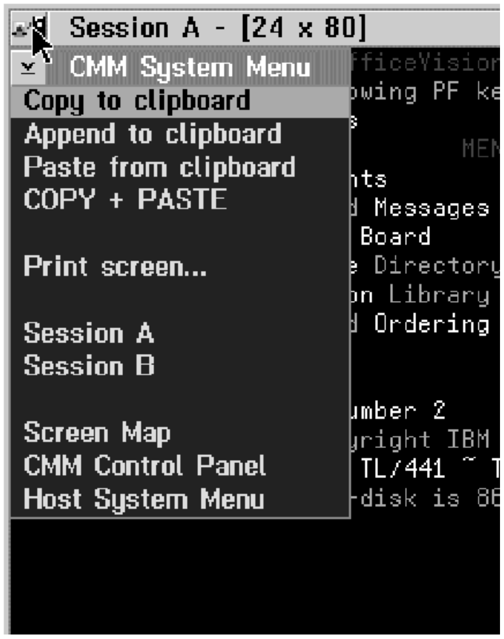


Figure 7. Host System Menu (OS/2 and Windows)

If you need to access the actual emulator system menu you can do so by selecting the *Host System Menu option*. This displays the standard host emulator system menu.

Clipboard Functions

All of the clipboard (cut and paste) operations that are normally provided by the host emulator are available when you use CM Mouse.

These functions operate as they normally would, except for “marking” text in a 3270 host window. To mark text in a 3270 window you must press the LEFT mouse button (as usual) and drag the mouse pointer a short distance before the operation actually starts. For example, to mark a block of text to cut or copy, press and hold the LEFT mouse button and move the mouse a short distance. You will notice that an expanding rectangle appears after the mouse has moved a short distance. The origin of the rectangle is the position of the mouse when you first pressed the button. When the rectangle encloses the text you want to cut or copy, release the button and the rectangle “snaps” to the nearest text boundaries. The text can now be cut or copied to the system clipboard by selecting the appropriate emulator options.

1. Position the mouse pointer at the starting character.
2. Press and hold the CTRL key.
3. Press and hold the left mouse button and drag the mouse until the desired lines are hilighted.
4. Release the left mouse button.

5. Release the CTRL key.

To paste text to the host session:

1. Press and hold the CTRL key.
2. Click the middle mouse button.
3. Release the CTRL key.

CM Mouse Screen Map Facility

OS/2

Before you can effectively use CM Mouse on your host application, you need to understand how CM Mouse has been customized for that specific application. The CM Mouse *Screen Map* facility is designed to help you discover how to use CM Mouse on a particular host application. It will show you where you can “click” on the host screen, and what the different mouse buttons do.

The idea behind the Screen Map is to identify what will happen when you click on various areas of a specific host screen. If “hot spots” have been defined to CM Mouse for that screen, they will be displayed as blocks of color. Each hot spot will be shown in a different color so that it can easily be seen. If you click in a block of color (hot spot), the Screen Map will show you what each mouse button will do in that area.

Displaying the Screen Map

OS/2

To display the Screen Map for a specific host screen, first enter the appropriate host commands to display the screen you want. Now locate the CM Mouse icon on the desktop and click on it. A pop-up menu will appear:

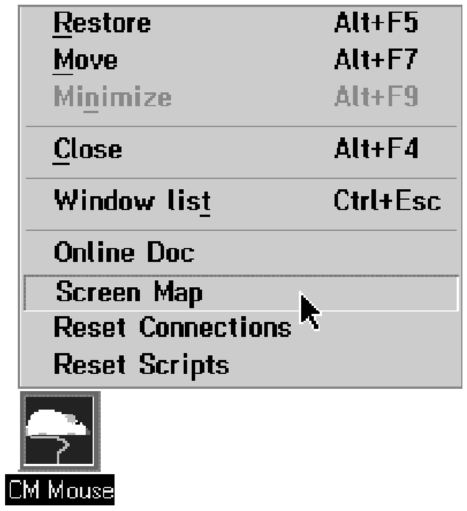


Figure 8. Selecting the Screen Map Option

Select the *Screen Map* option and a menu will be shown listing the currently active host sessions:

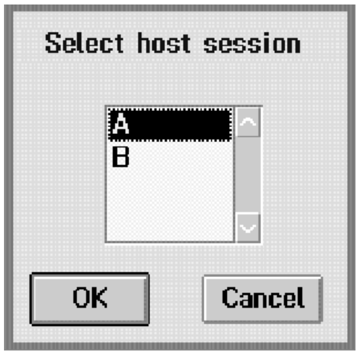
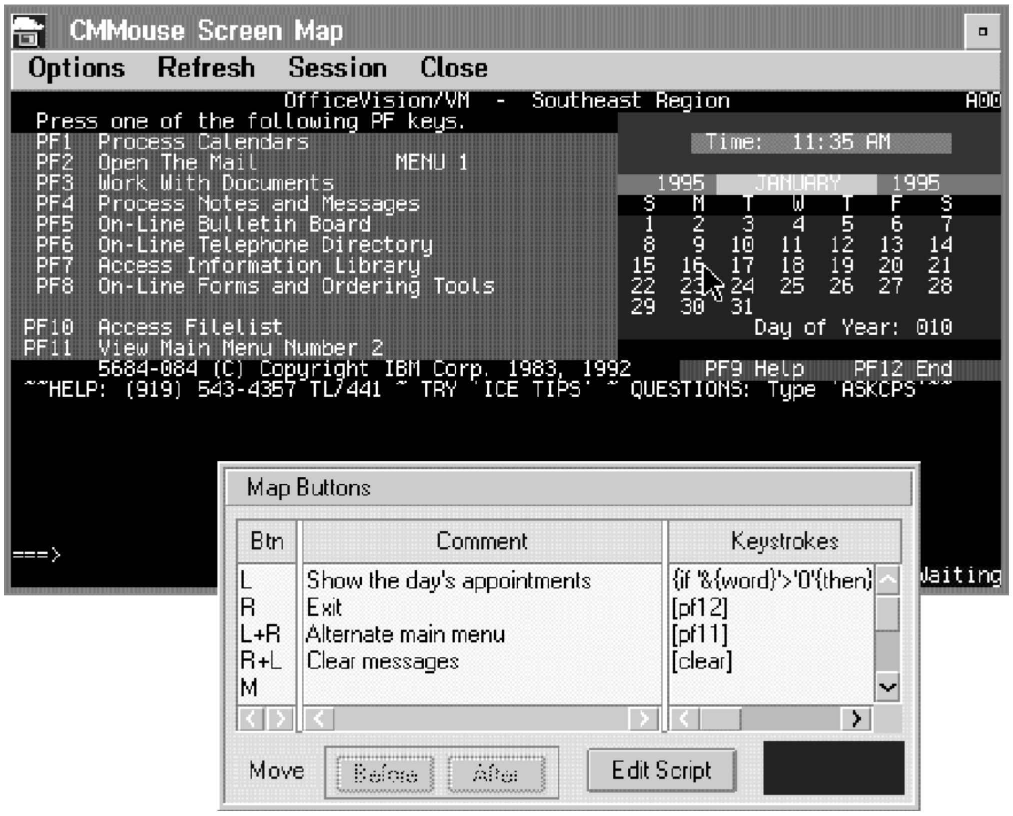


Figure 9. Selecting a Host Session

Select the host session which contains the screen you want and then click on the **OK** button. Two windows will be displayed (see Figure 10 on page 22). The first is the Screen Map main window. It contains a duplicate image of the host screen and an action bar with several items on it. The second window is the **Map Buttons** window. It contains a three-column list of data and several buttons. You will use these two windows to learn how CM Mouse was customized for the specific host screen being shown.



```
                    OfficeVision/VM  -  Southeast Region                    A00
 Press one of the following PF keys.
 PF1  Process Calendars                                 Time:  11:35 AM
 PF2  Open The Mail          MENU 1
 PF3  Work With Documents                         1995    JANUARY    1995
 PF4  Process Notes and Messages                   S   M   T   W   T   F   S
 PF5  On-Line Bulletin Board                       1   2   3   4   5   6   7
 PF6  On-Line Telephone Directory                  8   9  10  11  12  13  14
 PF7  Access Information Library                  15  16  17  18  19  20  21
 PF8  On-Line Forms and Ordering Tools            22  23  24  25  26  27  28
                                                  29  30  31
PF10  Access Filelist                                   Day of Year: 010
PF11  View Main Menu Number 2
      5684-084 (C) Copyright IBM Corp. 1983, 1992     PF9 Help    PF12 End
~~HELP: (919) 543-4357 TL/441 ~ TRY 'ICE TIPS' ~ QUESTIONS: Type 'ASKCPS'~~



===>                                                                    Waiting
```

Btn	Comment	Keystrokes
L	Show the day's appointments	{if '&{word}'>'0'{then}
R	Exit	[pf12]
L+R	Alternate main menu	[pf11]
R+L	Clear messages	[clear]
M		

Figure 10. Screen Map Windows

Using the Screen Map

OS/2

The main screen map window shows a duplicate image of the host screen with the text written in white, and the background in black. Different hot spots (mouse-sensitive) areas of the screen are indicated with different background colors. Each block of color indicates a single hot spot. Hot spots may span several rows and columns and are usually rectangular in shape. There may be any number of different hot spots shown (or none) depending on how CM Mouse has been customized for the particular host screen being shown.

The **Map Buttons** window shows a multicolumn list containing one line for each host button (including button chords and double-click buttons). Each line of the list shows the button type (first column), a description of what the button does (second column), and the actual host keystrokes associated with the button (third column).

To learn how CM Mouse is configured for the host screen being displayed, position the mouse cursor over one of the hot spots and press the LEFT mouse button. The **Map Buttons** window will be updated to show what the mouse buttons will do in that specific area (hot spot). A color bar at the bottom of the **Map Buttons** window will change to match the color of the hot spot you selected.

For example, in Figure 10 on page 22 the LEFT button was clicked with the pointer in the calendar display area. The **Map Buttons** window shows what each mouse button does in that area; the LEFT button will display a calendar entry for the day under the mouse pointer, the RIGHT button will exit OfficeVision, and the double-click buttons will display the second OfficeVision main menu. The center column of the **Map Buttons** window contains a brief text description of the function the button performs. The rightmost column shows the actual keystroke sequence used to implement that function. Most users can ignore the rightmost column; it will be of interest only if you are creating button definition script files to customize CM Mouse.

By pointing to different places on the host screen image in the main Screen Map window and then clicking the LEFT button, you can discover what functions are assigned to the mouse buttons.

To remove all the Screen Map windows, click on the *Close* menu bar item. Each of the remaining menu bar options is described in the following sections.

Options->Button Window
This option hides/shows the **Map Button** window.

Options->Trace Window
This option hides/shows the **Map Trace** window. This window is of primary interest to those developing or debugging CM Mouse scripts. See “Map Trace Window” on page 24.

Options->Tracking Mode
When enabled, Tracking Mode automatically causes the **Map Buttons** and **Map Trace** windows to be updated as the mouse pointer is moved around the main Screen Map window. This means that the comments in the Map Buttons window will be automatically updated without having to click in each area identified by a block of color.

Options->Auto Update
When enabled, Auto Update causes the contents of the map windows to be updated automatically whenever the host screen changes. Note that the map windows are updated only when CM Mouse recognizes a new (and different) host screen as determined by the SCREEN statements in the CM Mouse script files. The map windows are not updated for insignificant host screen changes.

Options->Primary Window on Top
All the map-related windows will ’resurface’ on top of all other windows when you click on any one of them (they come to the top as a group). This option controls which of the map windows will be on top of the others when the map windows are resurfaced by clicking on one of them. When this option is enabled, the main screen map window will always overlay the **Map Buttons** and **Map Trace** windows. When disabled, the main screen map will always be behind the other map windows.

Options->Fonts
This option will display the standard OS/2 font selection dialog. This dialog may be used to select a different font for the main map window. Note that only fixed-pitch fonts will be available for selection (no proportional fonts).

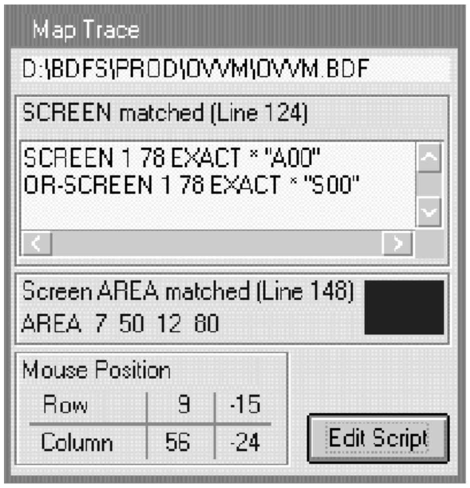
Refresh
This option can be used to cause an immediate update of all the map windows from the current host screen data. When the **Options->Auto Update** option is enabled this is usually not required.

Session
This option can be used to change the session to be mapped.

Close This option will close all the map windows.

Map Trace Window

If you will be creating or customizing CM Mouse button definitions, the Map Trace window displays technical information about the current host screen that you will find very useful. The Map Trace window is enabled and disabled with the *Options->Trace Window* menu option on the main map window. A typical Map Trace window is shown in Figure 11



```
Map Trace
D:\BDFS\PROD\OVVM\OVVM.BDF
SCREEN matched (Line 124)
SCREEN 1 78 EXACT * "A00"
OR-SCREEN 1 78 EXACT * "S00"
Screen AREA matched (Line 148)
AREA  7  50  12  80
Mouse Position
  Row       9   -15
  Column   56   -24
```

Figure 11. Map Trace Window

The Map Trace window contains three main sections:

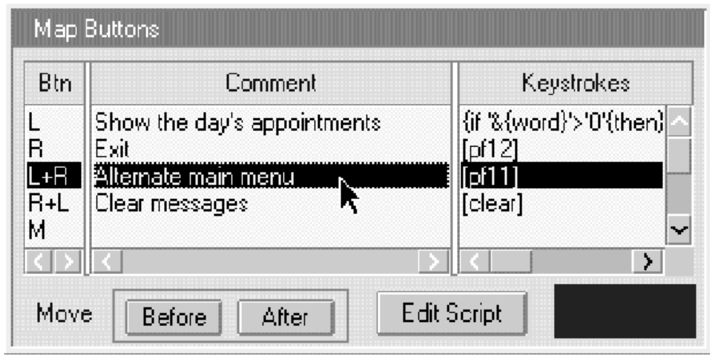
- At the top is the name of the BDF file containing the currently matched SCREEN statement (if there is one). If the name extends off the right edge of the window, select the text with the mouse and use the cursor movement keys to scroll the text right and left. Immediately following the file name is a match/nomatch message. If a SCREEN statement is matched, the line number of the matching SCREEN statement in the BDF file is shown. Also shown is the actual matching SCREEN statements. If the SCREEN statement included AND-SCREEN or OR-SCREEN statements, they will all be shown in the list box.
- The middle section contains information about the currently matched AREA statement (if any). The first line is a match/nomatch message. If there is an AREA match, the line number of the AREA statement in the BDF file is shown along with the actual AREA statement and a color block which matches the color of the area in the main Screen Map window.
- The bottom section of the window shows information about the current mouse position in the Screen Map main window. The mouse row and column position is shown in absolute terms (the first column of numbers), and relative to the end of the screen (the second column). This information can be used when building BDF files to easily determine the correct parameters for SCREEN and AREA statements.

  This section also contains an **Edit Script** button which, when selected, invokes the editor on the file named at the top of the window. If the BDF file is not in the first directory in the BDF/MMM path you will be given the option of copying the file to that directory.

As with the Map Buttons window, this will only be updated when the LEFT button is clicked in the Screen Map main window unless tracking mode is enabled. When tracking mode is enabled, the Map Trace window updates automatically as the mouse is moved in the Screen Map main window.

Customizing the Map Buttons Window

The Map Buttons window (see Figure 12) can be customized to show the information that is of most interest to you. You can rearrange the order of the buttons shown in the multicolumn list, and you can move the divider bars between the columns to display more information in one column or another.



Btn	Comment	Keystrokes
L	Show the day's appointments	{if '&{word}'>'0'{then}
R	Exit	[pf12]
L+R	Alternate main menu	[pf11]
R+L	Clear messages	[clear]
M		

Figure 12. Map Buttons Window

To reorder the list use the right mouse button to drag and drop the rows into the order you want. You can move entries until the list shows the buttons of most interest to you at the top.

To move the divider bars of the multicolumn list box, position the mouse over either of the dividers, press and hold the left button, and drag the bar to its new position. This can be used to widen any column of the multicolumn list (making an adjacent column narrower).

Chapter 6. CM Mouse BDF Script Files

A CM Mouse Button Definition File (BDF) script is a plain-text file which resides on the workstation or file server. It is read by CM Mouse when CM Mouse is initialized and when the **Reset->Scripts** control panel option is used. There may be any number of BDF script files on the workstation. CM Mouse may be setup to read all of them or only those needed for specific host applications. BDF scripts can be modified or created with any workstation text editor.

One special BDF script file is known as the *primary* script file (by default named CMMOUSE.BDF). All other script files are loaded through INCLUDE statements in the primary BDF script. Scripts loaded through INCLUDE statements may themselves load more scripts. The primary BDF file usually consists of only default button definitions and INCLUDE statements (in general, one INCLUDE statement for each host application).

A BDF script for a specific host application contains information CM Mouse needs to properly interpret mouse clicks on the screens of that application. In particular, the script tells CM Mouse:

1. *how to recognize different screens of the application,*

   Screen recognition is is one of the most power aspects of CM Mouse. By recognizing specific screens, CM Mouse can be truly context-sensitive; hot spots can be different on each host screen and mouse clicks can be tied to specific areas of specific screens.

   Screen recognition is accomplished through the use of SCREEN statements in an application's BDF script file.

2. *where the hot spots are on each screen,*

   Hot spots define particular areas of the host screen that have unique purposes. For example, a hot spot might be defined over a PF key list so that clicking anywhere in that list causes the appropriate PF key to be sent to the host. This concept of hot spots is quite different from the static 'hot spot' many emulators support which assigns fixed functions to the mouse buttons based on text which appears under the cursor.

   Hot spot areas are defined in an application BDF script file with the AREA statements.

3. *what to do when the user clicks in a particular hot spot on a particular screen.*

   Once screens have been recognized and hot spots established for each screen, BUTTON statements define what sequence of actions is to be performed if the user clicks a particular button in a particular hot spot. The sequence of actions may include sending keystrokes to the host, reading data from the host screen, or running workstation programs.

The SCREEN, AREA, and BUTTON statements are the key parts of the BDF script files. The syntax and format of each BDF script statement is described in detail in the following sections.

See Chapter 14. Sample Button Definitions and Menus, for a step-by-step example of building a button definition file for a sample host application.

Button Definition File General Syntax

All statements in a BDF script file must conform to the following general syntax rules:

- Any line that starts with an asterisk (*) in column 1 is taken to be a comment and is ignored. Blank lines are also ignored. Comment lines cannot be continued with the line continuation character (that is, they are not continued even if they end with “+”).
- Whenever a statement contains a double-quoted (″) string, the closing quote is optional if the string is not followed by a comment. It is required if the string is to contain trailing blanks or a comment. Comments which follow quoted string may not themselves contain a double quote character. If the string itself is to contain a double quote, the closing quote is required and a comment is not allowed to follow.
- All non-comment lines must begin with one of the following statement-type keywords:

  ```
  BUTTON
  SCREEN
  AND-SCREEN
  OR-SCREEN
  AREA
  SET
  INCLUDE
  DOIF
  ENDIF
  MAP
  DEFINE
  ENDDEFINE
  DRAG
  DROP
  ```

  Each statement type is explained in detail in the following sections.
- Statements can be in any combination of upper and lower case characters. Note, however, that double-quoted strings will be used exactly as they appear.
- Statements can start in any column, and parameters can be separated by any number of blanks. (Note however that comments must have an asterisk in the first position).
- No single line can be more than 4000 characters long, and no statement can exceed 4000 characters after continuation.
- Statements can be continued from one line to the next, as long as they do not exceed a total length of 4000 characters. To continue a statement, the line must end in a “+” (plus) character. A line ending in this character is continued to the next line. All *leading* blanks on the next line are removed before the lines are concatenated. This allows continued lines to be indented for readability if desired. For example:

  ```
  * ++++++ This is a comment, and is not continued ++++++++
  BUTTON LEFT   "This is a button definition +
                 string which is quite long."
  BUTTON RIGHT  "This is +
                 very +
                 very +
                 very +
                 redundant."
  ```

Button Definition File Structure

The general form of a CM Mouse BDF script file is shown in Figure 13.
The indentation shown above is not required, but is useful for easily viewing the

```
SCREEN <a>        (recognize screen <a>)

  BUTTON ...      (default buttons for screen <a>)
  BUTTON ...
  ...

  AREA <1>        (define hotspot <1> on screen <a>)

    BUTTON ...    (define buttons for hotspot <1>)
    BUTTON ...
    ...

  AREA <2>        (define hotspot <2> on screen <a>)

    BUTTON ...    (define buttons for hotspot <2>)
    BUTTON ...
    ...

SCREEN <b>        (recognize screen <b>)

  BUTTON ...      (default buttons for screen <b>)
  ...

  AREA <3>        (define hotspot <3> on screen <b>)

    BUTTON ...    (define buttons for hotspot <3>)
    BUTTON ...
    ...
```

Figure 13. General Structure of a Button Definition File

relationship between BUTTON, AREA, and SCREEN statements.

In general, each unique screen of the host application will have one SCREEN statement in the script file. Following the SCREEN statement is a list of default buttons for that screen (this defines the actions to be taken if the user clicks outside all hot spots). Following the default BUTTONs are AREA statements. Each AREA defines a single rectangular hot spot of the host screen. The BUTTON statements following an AREA statement define the actions to be taken when the user clicks a particular button in that specific AREA (hot spot).

An AREA is terminated by the next AREA or SCREEN statement. A SCREEN is terminated by the next SCREEN statement.

The Primary BDF Script

The *primary* BDF script file is the first script file read by CM Mouse. The default name for this file is CMMOUSE.BDF (see “Chapter 4. Configuring CM Mouse” on page 9). It is unique in several ways:

1. It is the first and only script file read directly by CM Mouse. Other script files are loaded by INCLUDE statements in the primary BDF script. CM Mouse does not search the disk for all script files; script files must be explicitly named in an INCLUDE statement of the primary BDF. Script files are loaded and processed in the order of their INCLUDE statements.

2. The primary BDF script file usually contains only INCLUDE statements for application-specific scripts. By convention it does not contain any SCREEN or AREA statements. Thus the primary BDF file acts only as a list of application scripts to be loaded and does not itself define any host screens or hot spots. Note that this is by convention only; technically the primary BDF is processed like any other BDF and may contain any valid BDF statement.
3. The primary BDF script file may have BUTTON statements before the first INCLUDE statement. These BUTTON definitions are referred to as the *global default* buttons. These BUTTON statements define the actions to be taken when the host screen is not recognized by any of the application scripts. (The current host screen does not match any SCREEN statement in any application BDF script). They are also used if the screen is recognized but has no BUTTON statement for the selected button.

   The global default buttons are usually defined to be some harmless action like {beep} which tells the user that the current host screen is not recognized by CM Mouse. Global default button definitions may only be specified in the primary BDF file and must appear before the first INCLUDE or SCREEN statement.

The simplest CM Mouse script configuration file might have no INCLUDE or SCREEN statements at all, only a set of global default buttons. For example, if you wanted to simply define the left button to act as a light-pen selection and the right button to be PF3 on all host screens, then the primary BDF file could simply consists of the following two lines:

```
BUTTON LEFT  "{seek}[crsel]"
BUTTON RIGHT "[pf3]"
```

Any button not explicitly defined will have no function. (The syntax of the BUTTON statement is described in “BUTTON Statement” on page 34.)

The global default buttons are also used when a SCREEN is matched but has no BUTTON definition for the button that was pressed. For example, given the following primary BDF script:

```
BUTTON LEFT "[pf1]"
BUTTON RIGHT "[pf2]"
SCREEN ...
  BUTTON LEFT "[pf3]"
```

If the SCREEN was matched successfully, then the LEFT button would produce [pf3] and the RIGHT button would produce [pf2]. If the screen was not matched, then the LEFT button would produce [pf1] and the RIGHT button would still produce [pf2].

Note

Global button definitions should be used with care. It is usually not possible to define a set of global definitions which are useful on all host applications. If you run a host application for which no BDF has been written, the global default buttons are used. If those button definitions make assumptions about how the host application works, those assumptions could be wrong in some cases.

For example, suppose the RIGHT button is defined in the global defaults to be "[pf3]" since most applications use that key to exit the menu. This will not work on host applications which use PF12 to exit the menu. If you press the RIGHT button on such an application (expecting to exit the menu), the PF3 key may have an undesirable effect.

It is best to define the default buttons as "{beep}" or some other harmless function to prevent unexpected results on undefined host screens.

Default Button Definitions for SCREEN Statements

Each SCREEN statement defines certain characteristics that CM Mouse will look for in host screens. If those characteristics match the current host screen, then the BUTTON statements within that SCREEN definition are used. If a particular BUTTON is not defined within a SCREEN definition, then the global default button definition is used. SCREEN definitions are matched with the host in sequential order (from top to bottom), and the first match found is used. If no SCREEN definition matches the current host screen, the global default button definitions are used.

Each SCREEN definition can contain any number of AREA definitions. An AREA defines a rectangular subarea ("hot-spot") of the host screen. If the mouse cursor lies within the AREA rectangle when a mouse button is pressed, the BUTTON definitions immediately following the AREA statement are used. Note that a given AREA definition is used only if its SCREEN was successfully matched. Within a SCREEN definition, AREA definitions are examined in sequential (top down) order. AREA rectangles may overlap, and the first AREA found containing the mouse cursor position will be used. If the mouse cursor does not lie within any AREA defined for the SCREEN, the SCREEN button defaults are used. If a particular button is not defined by an AREA, the SCREEN default button definition is used; if there is no SCREEN default button definition, the global default definition is used.

The CM Mouse button definition file defines a hierarchy of SCREENs and AREAs (as illustrated by the indentation in Figure 13 on page 29). The file can contain any number of SCREEN definitions, and each SCREEN definition can contain any number of AREA definitions. This provides the capability to perform almost any interaction with a host application based on the position of the mouse cursor when a button is pressed.

Button Definition File Statements

The following section describes the statements that can appear in a CM Mouse Button Definition File (BDF) script.

SCREEN Statement

The SCREEN statement is used by CM Mouse to identify particular host screens. This is done in two ways. First, the screen must appear on the specified host session (a, b, c...), and, secondly, it must contain a specified string of characters in a particular location. Techniques are provided to specify “any” host session (the screen is recognized no matter which session it appears on), and the character string can be defined as follows:

- In a specific row/column location
- Anywhere on a particular line
- Anywhere on the screen.

The location can also be specified relative to the size of the host screen, so that SCREEN definitions will work even on different sizes of host screens (24x80, 32x80, and so on).

The format of the SCREEN statement is:

```
►►──SCREEN──Row──Column──┬─EXACT──┬──┬─A────┬──"──FindString──"──┬─────────┬──►◄
                         ├─LINE───┤  ├─B────┤                     └─Comment─┘
                         └─SCREEN─┘  ├─C────┤
                                     ├─...──┤
                                     ├─3270─┤
                                     ├─5250─┤
                                     └─*────┘
```

Row and *Column* specify the starting row and column of the string search (the upper left corner of the screen is row 1 column 1). These values may be specified as zero or negative values to indicate an offset from the end of the screen (*Row*=0 is the last line of the screen, *Row*=-1 is the second from the last. *Column*=0 is the last column of the screen, -1 is the second from the last).

Following the row and column is the *search mode* which indicates how the string is to be searched for on the host screen. There are three allowed values:

EXACT
The string must appear at the exact row and column.

LINE The string may appear anywhere after the specified row and column, on the same line.

SCREEN
The string may appear anywhere after the specified row and column, to the end of the screen.

After the search mode is the *session* parameter which indicates the host session on which the SCREEN will be recognized. This parameter allows the matching to be restricted to certain host sessions. The session to be matched may be specified as:

Session Parameter	Match
A, B, C, ...	Match only on session (single character session ID)
*	Match on any host session
3270	Match on any 3270 host session

Session Parameter	Match
5250	Match on any 5250 host session

The *session* is usually coded as “*” so that the screen will be recognized no matter which host session it appears on. The screen matching should only be restricted to a particular session when it is known that the screen of interest will only appear on that session.

For users who have both 3270 and 5250 hosts, restricting SCREEN matches to a specific type of host can help improve CM Mouse performance.

AND-SCREEN Definition Statement

Any number of AND-SCREEN statements can follow a SCREEN statement. They are used to further qualify the match requirements of the SCREEN. The SCREEN conditions must match, and all the AND-SCREEN conditions must match for the host screen to be recognized. For example, consider the following script:

```
SCREEN      1 1 exact * "TITLE"
AND-SCREEN  2 1 exact * "SUB-TITLE"
AND-SCREEN  3 1 exact * "===>"
BUTTON LEFT...
...
```

This screen is recognized only if `TITLE` appears in the upper left corner of the screen *and* the second line starts with `SUB-TITLE`, *and* the third line starts with the characters “===>”. If all those conditions are met, the AREAs and BUTTONs following the SCREEN statements are used. Otherwise, the search for a screen match continues with the next SCREEN statement in the script.

This can be useful to help distinguish host screens which are similar but need to be processed differently.

Consider another example:

```
SCREEN     1 1 exact * "Main Title"
AND-SCREEN 2 1 exact * "Submenu 1"
BUTTON LEFT ...

SCREEN     1 1 exact * "Main Title"
AND-SCREEN 2 1 exact * "Submenu 2"
BUTTON LEFT ...

SCREEN     1 1 exact * "Main Title"
BUTTON LEFT ...
```

In this example, the first SCREEN is recognized if `Main Title` appears in the upper left corner, and `Submenu 1` is below it. The second SCREEN is recognized if `Submenu 2` is below it. If `Main Title` is in the upper left and neither `Submenu 1` nor `Submenu 2` is below it, then the third screen is recognized.

An AND-SCREEN statement always applies to the nearest previous SCREEN statement. AND-SCREENs and OR-SCREENs cannot both be applied to the same SCREEN statement. For example, the following is not valid:

```
SCREEN     1 1 exact * "My Title"
AND-SCREEN 2 1 exact * "Subtitle 1"
OR-SCREEN  2 1 exact * "Subtitle 2"
```

OR-SCREEN Definition Statement

Any number of OR-SCREEN statements can follow a SCREEN statement. They are used to further qualify the match requirements of the SCREEN. The SCREEN conditions can match, or any of the OR-SCREEN conditions can match for the host screen to be recognized. For example, consider the following script:

```
SCREEN     1 1 exact * "MENU 1"
OR-SCREEN  1 1 exact * "MENU 2"
OR-SCREEN  1 1 exact * "MENU 3"
OR-SCREEN  1 1 exact * "MENU 4"
BUTTON LEFT...
```

This screen is recognized if `MENU 1` appears in the upper left corner, or `MENU 2` appears in the upper left corner, or `MENU 3`, or `MENU 4`. This is very useful for defining the same hot spots and button definitions for a set of similar host screens.

An OR-SCREEN statement always applies to the nearest previous SCREEN statement. OR-SCREENs and AND-SCREENs cannot both be applied to the same SCREEN statement. For example, the following is not valid:

```
SCREEN     1 1 exact * "My Title"
AND-SCREEN 2 1 exact * "Subtitle 1"
OR-SCREEN  2 1 exact * "Subtitle 2"
```

BUTTON Statement

This statement is used to identify what keystrokes (or special functions) are to be associated with a particular button. The format of the statement is:

```
►►──BUTTON───┬─LEFT─────────┬─"──ButtonDefinition──"──┬─────────┬─────────────►◄
             ├─RIGHT────────┤                         └─Comment─┘
             ├─MIDDLE───────┤
             ├─LEFT+RIGHT───┤
             ├─LEFT+MIDDLE──┤
             ├─RIGHT+LEFT───┤
             ├─RIGHT+MIDDLE─┤
             ├─MIDDLE+LEFT──┤
             ├─MIDDLE+RIGHT─┤
             ├─DBLLEFT──────┤
             ├─DBLRIGHT─────┤
             └─DBLMIDDLE────┘
```

The LEFT, RIGHT, or MIDDLE parameters define the operation of that button when “clicked” (pressed and released) without any other buttons. The LEFT+RIGHT button defines the operation of first pressing and holding the LEFT button and, while it is down, pressing and releasing the RIGHT button. (This is referred to as a button “chord”.) In general, the button named first is the button to be pressed and held while the second button is pressed and released. For a two-button mouse, the button names using “MIDDLE” cannot be used. (They can be defined, but will never be used.) The DBLLEFT, DBLRIGHT, and DBLMIDDLE buttons define the double-click operation of the buttons.

If you are using a three-button mouse, then up to 12 different button combinations can be used (3-way button clicks are not recognized by CM Mouse; therefore, you cannot define “LEFT+RIGHT+MIDDLE”). For a two-button mouse, 6 different button combinations can be used.

OS/2

CM Mouse depends on the OS/2 Presentation Manager (PM) to get mouse information, so the Presentation Manager must be set up to recognize all three buttons of a three-button mouse.

ButtonDefinition is the sequence of keystrokes to be sent to the host when the named button is pressed. It can consist of simple characters (which are typed onto the host screen at the current host cursor location exactly as given), host PF keys, almost any keyboard key, or special CM Mouse control keywords. All of these are described in Chapter 8. Button Definitions. Note that the quoting rule as described in “Button Definition File General Syntax” on page 28 applies to the *ButtonDefinition* string.

The *comment* should be a brief (one line) description of what the button does.

Example 1:

```
BUTTON LEFT "abc def[enter]" ...and this is a comment
```

This would define a single click of the LEFT button as the keystrokes `abc def` followed by the ENTER key.

Example 2:

```
BUTTON RIGHT "&{popup mymenu}"
```

This would define a single click of the RIGHT button as the CM Mouse substitution word `popup` which causes a menu to be displayed on the host screen. (See Chapter 8. Button Definitions, for details on CM Mouse substitution and control words.)

AREA Statement

The AREA statement is used by CM Mouse to subdivide a particular SCREEN into rectangular areas (hot spots). Any number of AREAs (or none) can be defined within a single SCREEN definition. If the mouse cursor does not lie within any AREA of a SCREEN definition, the SCREEN default button definitions are used. If it does lie within an AREA, then the BUTTON definitions immediately following the AREA statement are used.

The rectangular AREA can be defined in terms of absolute row and column numbers, or it may be defined relative to the size of the host screen. This allows AREAs to be defined relative to the end of the screen and still work on different sized host sessions (such as 24x80, or 32x80).

The format of the AREA statement is:

```
►►──AREA──StartRow──StartColumn──EndRow──EndColumn──┬─────────┬──────────────►◄
                                                    └─Comment─┘
```

StartRow and *StartCol* specify the location of the upper-left corner of the rectangular area (row 1 column 1 is the upper left corner of the screen), and *EndRow EndCol* specify the lower right corner of the area. Any of the values can be specified as zero or negative to be offset from the end of the screen. For example, a row number of 0 is the last line of the screen, -1 is the second to last, and so on. A column of 0 is the last column of the screen, -1 is the second to last, and so on. The starting row must be less than the ending row, and the starting column must be less than the ending column. (CM Mouse does not check this. If one of the ending values is less than the starting value, then a *null* area is defined, and it will never be used.)

Example:

```
AREA 2 1 2 65 -- Command area of ISPF editor
```

The preceding AREA would define the second line of the screen, from columns 1 through 65. Another example:

```
AREA -1 1 0 0 -- Two PFkey lines at bottom of screen
```

This AREA contains the last two lines of the screen. Note that this will always be the last two lines, regardless of the size of the host screen (for instance, 24x80, or 32x132).

SET Statement

A SET statement can appear anywhere in a button definition file. The format is:

```
►►──SET──VariableName──┬─────────┬──"──VariableValue──"──┬─────────┬──────────►◄
                       └─LITERAL─┘                       └─Comment─┘
```

The *VariableValue* can be formed from explicit characters and from substitutions. (Note that only *presubstitutions* are allowed in SET statements, no runtime substitutions are allowed.) If the **LITERAL** option is specified then no substitutions are done and the *VariableValue* is used exactly as it appears.

SET statements are only processed when the BDF file in which they appear is read. BDF files are read only during initialization and when the **Reset->Reload scripts** option is selected on the Control Panel.

SET statements in BDF scripts may not contain any substitution which requires information from or about any host session. Thus it may not contain any of the following keywords:

```
 &{chars} &{word} &{num} &{hrow} &{hcol} &{mrow}
&{mcol}
 &{popup} &{sid} &{srow} &{scol} &{?...}
```

Note there is no such restriction on SET statements which appear in a menu (.MMM) file.

SET statements have many uses. For example, they can be used to customize CM Mouse for a particular user. For example, a BDF script might be as follows:

```
SET UserID     "TJSMITH"
SET SystemID   "VM18"
SET KeyHelp    "[pf9]"
SET KeyExit    "[pf12]"
SET KeyDelete  "[home]erase[enter]"
...
...
BUTTON LEFT  "&{var KeyHelp}"
BUTTON RIGHT "&{var KeyExit}"
...
BUTTON LEFT  "sendfile abc to &{var UserID} at &{var SystemID}[enter]
...
```

In this example, you can customize the operation of the script simply by changing the SET statements. For example, if you have changed the host application to exit on PF3 instead of PF12, then you can simply change the setting of the *KeyExit* variable. User names might also be SET in one place so that they can be changed easily. This allows a BDF scripts to be customized in just one place so you do not have to read and analyze the entire script file to change it.

See “Chapter 11. CM Mouse Variables” on page 105 for more information on the use of variables.

INCLUDE Statement

An INCLUDE statement can appear anywhere in a button definition file. The format is:

```
►►──INCLUDE──┬─────────────────┬──FileName──┬─────────┬──────────────────────────►◄
             └─RelativePath\───┘            └─Comment─┘
```

The optional *relative path* name should not include a drive letter or file name extension. The BDF/MMM path specified in the CM Mouse configuration will be used to locate the named script file (see “Chapter 4. Configuring CM Mouse” on page 9). If the configuration specifies multiple BDF/MMM directories they are searched in order for the file name specified. The file name extension should **not** be specified (it will always be “.BDF”).

The file is “included” into the button definition file at the point of the INCLUDE statement. It is processed exactly as if the lines of the file appeared in place of the INCLUDE statement. An included file may itself contain INCLUDE statements, but such nesting is limited to four levels.

This statement is often used in the primary BDF script to load the scripts for each specific host application. This statement can also be used to break up large button definition files into logical sections.

Example:

```
INCLUDE filelist -- Button defs for FILELIST
```

This would include the file FILELIST.BDF into the current button definition file. Each directory specified in the BDF/MMM configuration list would be searched until the file was found.

DOIF/ENDIF Statements

These statements allow blocks of script to be excluded from processing based on a specified condition. The excluded script is not read or processed in any way. The statement format is:

```
►►──DOIF──'string1' ──┬─=───┬──'string2' ──────────────────────────────►◄
                      ├─==──┤
                      ├─!=──┤
                      ├─!==─┤
                      ├─<───┤
                      ├─>───┤
                      ├─<=──┤
                      ├─>=──┤
                      ├─=#──┤
                      └─!=#─┘
```

```
►►──ENDIF──────────────────────────────────────────────────────────────►◄
```

Any BDF statements between the DOIF and ENDIF statements are skipped if the expression is false. Skipped statements are not processed in any way and are not evaluated for syntax errors. If the expression is true, the statements between the DOIF and ENDIF are read and processed normally.

The comparison operators are the same as those allowed in the {if...} keyword (see page “{if <condition> {then}script1{else}script2}” on page 59). The strings may be formed by any *presubstitution* CM Mouse keywords and/or literal strings. Runtime substitutions are not allowed.

NOTE

The DOIF conditions are evaluated when the BDF file is read (usually when CMMouse is first started). In general they will be evaluated *before* any connection is made to a host session. Thus the DOIF statement must not contain any CM Mouse substitution requiring information from or about any host session. For example, it must not use any of the following keywords:

&{chars} &{word} &{num} &{hrow} &{hcol} &{mrow} &{mcol} &{popup} &{sid} &{srow} &{scol} &{?...}

The DOIF/ENDIF statements may be nested.

Example:

```
DOIF '&{var SYSTEM_ENV}' = 'DOS'
  ...stuff only for DOS...
ENDIF

DOIF '&{var SYSTEM_ENV}' != 'DOS'
  ...stuff for non-DOS environments...
ENDIF
```

In general the DOIF/ENDIF statements can be used to make a script usable on any platform, even if it uses platform-specific features. For example, the REXX functions are available only on OS/2, so a script could be written as follows:

```
DOIF '&{var SYSTEM_ENV}' = 'OS2'
  button left "&{rexx...}"
ENDIF
DOIF '&{var SYSTEM_ENV}' != 'OS2'
  button left "{?Not supported.}"
ENDIF
```

MAP Statement

A MAP statement can appear anywhere in a button definition, but usually appears only once at the top of the script file. The syntax of the MAP statement is:

```
►►──MAP───┬─OFF─────────────┬──┬─────────┬──────────────────────────────────────►◄
          ├─CHORD TO DCLICK─┤  └─Comment─┘
          └─DCLICK TO CHORD─┘
```

The default for all BDF files is *OFF*. Each BDF file which is to have a mapping must explicitly have a MAP statement before the first BUTTON statement.

The purpose of the MAP statement is to simplify the writing of BDF files which support the interchangeable use of button chords and double-clicks. Some users tend to prefer the “chord” style of button pressing, and some prefer to use double-clicks. It is best to construct a BDF such that the user can choose which style to use and still have access to all the assigned button functions.

To do this manually would require duplicate BUTTON statements for the chord and double-click definitions:

```
  BUTTON LEFT+RIGHT "[pf7]" Scroll down
  BUTTON DBLLEFT    "[pf7]" Scroll down
  BUTTON RIGHT+LEFT "[pf8]" Scroll up
  BUTTON DBLRIGHT   "[pf8]" Scroll up
```

The above sample defines two sets of buttons for scrolling; the left/right chord combinations, and the left/right double-click buttons. Writing a script in this manner would allow either chords or double-clicks to be used to perform the same functions.

Writing an extensive script with duplicate definitions like that above can be tedious and error prone because each button function has to be defined twice (once for a chord, and once for a double-click). CM Mouse provides a means for the BDF to be written with just one set of definitions and have them automatically mapped into the other set. You can write either chord definitions or double-click definitions, and with an appropriate MAP statement CM Mouse will automatically map one to the other.

By *not* using a MAP statement, a script can be written to take full advantage of both chords and double-clicks to provide the maximum mouse function.

The MAP statement is used to specify the mapping to be used for the BDF file in which the MAP statement appears. MAP statements take effect only within the physical BDF file in which they appear -- they are not inherited by INCLUDE files.

Each of the mapping options is described in the following sections.

MAP CHORD TO DCLICK

When this mapping is in effect, each BUTTON statement specifying a chord will also automatically define the associated double-click BUTTON. For example, the following:

```
MAP OFF
...
BUTTON LEFT+RIGHT "x1"
BUTTON DBLLEFT "x1"
```

is exactly equivalent to:

```
MAP CHORD TO DCLICK
...
BUTTON LEFT+RIGHT "x1"
```

When this mapping is in effect, defining a chorded button is equivalent to defining the chord button and the associated double-click button. The associations are:

```
LEFT+RIGHT      --->      DBLLEFT
RIGHT+LEFT      --->      DBLRIGHT
MIDDLE+LEFT     --->      DBLMIDDLE
```

This mapping can be used to make double-clicking usable on BDFs written for a prior release of CM Mouse which did not support double-click buttons. The existing BDFs can be used without having to manually recode the BDFs to include the BUTTON DBLxxxx statements. Using the “MAP CHORD TO DCLICK” statement will automatically assign the same functions to the double-click buttons as are assigned to the chord buttons.

This automatic mapping can be overridden by explicitly defining the function of a double-click button. For example:

```
MAP CHORD TO DCLICK
...
BUTTON LEFT+RIGHT "[pf4]"
BUTTON DBLLEFT    "[pf7]"
```

Normally the MAP statement would cause the DBLLEFT button to be assigned the same function as LEFT+RIGHT. However, since an explicit definition is given for DBLLEFT, that value (“[pf7]”) is used instead. Note that the BUTTON DBLLEFT statement must appear *after* the BUTTON LEFT+RIGHT statement in order to override the mapping.

MAP DCLICK TO CHORD

When this mapping is in effect, each BUTTON statement which specifies a double-click will also automatically define the associated chord BUTTON. For example, the following:

```
MAP OFF
...
BUTTON DBLLEFT "x1"
BUTTON LEFT+RIGHT "x1"
```

is exactly equivalent to:

```
MAP DCLICK TO CHORD
...
BUTTON DBLLEFT "x1"
```

This allows a BDF to be written just in terms of double-click buttons, and still be used with button chords. When this mapping is in effect, defining a double-click button is equivalent to defining the double-click button and the associated chord button. The associations are:

```
DBLLEFT       --->     LEFT+RIGHT
DBLRIGHT      --->     RIGHT+LEFT
DBLMIDDLE     --->     MIDDLE+LEFT
```

This automatic mapping can be overridden by explicitly defining the function of a chord button. For example:

```
MAP DCLICK TO CHORD
...
BUTTON DBLLEFT    "[pf7]"
BUTTON LEFT+RIGHT "[pf4]"
```

Normally the MAP statement would cause the LEFT+RIGHT button to be assigned the same function as DBLLEFT. However, since an explicit definition is given for LEFT+RIGHT, that value ("[pf4]") is used instead. Note that the BUTTON LEFT+RIGHT statement must appear *after* the BUTTON DBLLEFT statement in order to override the mapping.

MAP OFF

This is used to disable button mapping. Any button statements which appear after this MAP statement will define only the button explicitly named.

DEFINE/ENDDEFINE Statements

This pair of statements is used to define large blocks of text which are not subject to the CM Mouse statement length limitations. Text inside of a DEFINE block is not limited to the maximum CM Mouse statement length and does not require CM Mouse line continuation characters.

The syntax of the DEFINE statement is:

```
►►──DEFINE──┬─NAMEMAP─┬──────────────────────────────────────────────►◄
            └─MapType─┘
```

```
►►──ENDDEFINE────────────────────────────────────────────────────────►◄
```

Currently the only supported DEFINE type is a NAMEMAP table. See "Chapter 10. Drag/Drop Features" on page 97 for a description of NAMEMAP tables.

DRAG Statement

This statement is used to associate a script sequence (keystrokes and functions) with a drag from the host screen to the OS/2 Workplace Shell or the Windows CM Mouse Drag/Drop file list window. The syntax of this statement is:

```
►►──DRAG──┬─FILE────┬──"──ButtonDefinition──"──┬─────────┬──────────────────►◄
          ├─PRINT───┤                          └─Comment─┘
          └─DISCARD─┘
```

Like the BUTTON statement, the DRAG statement is used within SCREENs and AREAs to give drag functionality to specific parts of specific host screens. If a particular area of a host screen has no DRAG statement associated with it (e.g. no DRAG statement in the AREA or SCREEN) then the user is not allowed to begin a drag from that position.

When the user drags from the host screen and drops on a WPS object or the Drag/Drop file list, the associated DRAG script is executed. The type of object that was dropped on determines which DRAG statement (FILE/PRINT/DISCARD) will be used. If the user drops on the WPS desktop, in a WPS folder or on the file list window of the Drag/Drop application, the DRAG FILE statement will be used. If the user drops on a WPS printer object the DRAG PRINT statement will be used. (The DRAG PRINT statement is not currently supported on Windows and will be ignored). If the user drops on a WPS shredder object or the shredder of the Drag/Drop application, the DRAG DISCARD statement will be used.

It is not necessary to support all types of drags in a given screen area. If a screen area does not have a particular type of DRAG statement associated with it, then drops on those types of objects will not be allowed. For example consider the BDF file:

```
SCREEN 1 1 exact * "File List"
BUTTON RIGHT "[pf3]"
AREA 2 1 -1 12 ------- List of files
  BUTTON LEFT "{seek}edit[enter]"
  DRAG FILE "{xfer ...}"
  DRAG DISCARD "{seek}delete[enter]"
```

In this sample a drag started on a file name would be allowed to drop on the WPS desktop, any WPS folder, and the system shredder (or in Windows, the Drag/Drop file list and the Drag/Drop shredder icon). Drop would not be allowed on a printer.

Note that what happens after a user drops is determined by the script which follows the appropriate DRAG statement. CM Mouse does not automatically assign any function to drag operations; the script author must determine the meaning of a drop on any particular type of object. In the example above, a drop on a shredder has the effect of sending a DELETE command to the host.

See “Chapter 10. Drag/Drop Features” on page 97 for a complete discussion of the DRAG/DROP functions and keywords.

DROP Statement

This statement is used to associate a script sequence (keystrokes and functions) with the drop of a OS/2 Workplace Shell object (or Windows Drag/Drop application object) on a particular place in the host screen. The syntax of this statement is:

```
►►──DROP──FILE──"──ButtonDefinition──"──┬─────────┬──────────────────►◄
                                        └─Comment─┘
```

Like the BUTTON statement, the DROP statement is used within SCREENs and AREAs to give drop functionality to specific parts of specific host screens. If a particular area of a host screen has no DROP statement associated with it (e.g. no DROP statement in the AREA or SCREEN) then the user is not allowed to drop an object in that position.

Currently CM Mouse supports only the dropping of FILE objects.

When the user drags a file from the WPS or Drag/Drop file list and drops on the host screen, the associated DROP script is executed. A drop is only allowed on the host screen where a DROP statement is associated.

For example consider the BDF file:

```
SCREEN 1 1 exact * "File List"
BUTTON RIGHT "[pf3]"
AREA 2 1 4 80 ------- Command area
  BUTTON LEFT "&{popup cmds}"
  DROP FILE "{xfer ...}"
```

In this sample a file object may (only) be dropped on lines 2, 3, and 4 of the host screen between columns 1 and 80. If a file object is dropped in that area the DROP FILE script will be executed.

Note that what happens after a user drops a file is determined by the script which follows the DROP FILE statement. CM Mouse does not automatically assign any function to drop operations; the script author must determine the meaning of a dropped file on the host screen.

See “Chapter 10. Drag/Drop Features” on page 97 for a complete discussion of the DRAG/DROP functions and keywords.

Chapter 7. CM Mouse Menu Files

CM Mouse allows you to define your own pop-up menus and modify any of those that are supplied as samples. These menus are invoked when an &{popup ...} substitution word appears in a button definition (see the description of this substitution word in “CM Mouse Substitution Words” on page 70). The file specified is read and the menu is displayed on the host screen with a *bounce bar* cursor. To select an option on the pop-up menu, move the mouse to position the bounce bar over the item you want and press the LEFT mouse button. To cancel the menu without selecting anything, press the RIGHT mouse button. Each menu file defines a single pop-up menu.

A menu file defines the appearance of a pop-up menu, including the text of each selectable item on the menu, menu colors, and a title line. A menu file also defines what is to be substituted when one of the items is selected. The substituted text can contain keystrokes to be sent to the host or CM Mouse control words including &{popup ...} to display another pop-up menu.

The format of the menu file is similar to the button definition file. Each line of the file contains a keyword followed by one or more parameters. Blank lines and lines with an asterisk (*) in column 1 are ignored. Lines cannot exceed 4000 characters in length after continuation. Lines can be continued by ending them in a “+” (plus) character. A line following a line ending in “+” is joined after removing all leading blanks.

Menu File Statements

Each non-comment line must contain one of the following statements:

```
PLACE
TITLE
COLORS
BAR
LINE
SET
DOIF
ENDIF
```

A menu file must have one or more LINE statements. If more than one PLACE, TITLE, COLORS, or BAR statements is found, only the last is used. Statements may appear in the menu file in any order, but the order of LINE statements define the order of selectable items in the menu.

Each statement type is described in the following sections.

PLACE

```
►►──PLACE──┬───────┬──┬──────────┬──────────────────────────────►◄
           ├─Row───┤  ├─Column───┤
           └─*─────┘  └─*────────┘
```

This statement determines where the pop-up menu is placed on the host screen. The row/column specified is the upper left corner of the pop-up menu. The upper left corner of the screen is row 1 column 1. If either value is specified as “*”, the current mouse cursor position plus 1 is used. Thus, “PLACE * *” causes the menu to display one row below and one column to the right of the mouse cursor. If there is no PLACE statement in the menu file, “PLACE * *” is used.

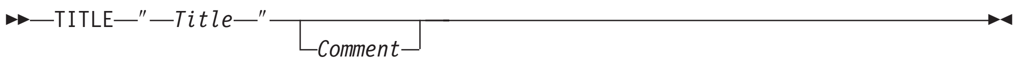
If the pop-up menu cannot fit on the screen in the location specified, it is fitted as close as possible to the edge of the screen. If no PLACE statement is specified in the menu file, “PLACE * *” is used.

Example:

```
place 1 *
```

This would cause the pop-up menu to appear starting on row 1, in the column immediately following the mouse cursor.

TITLE

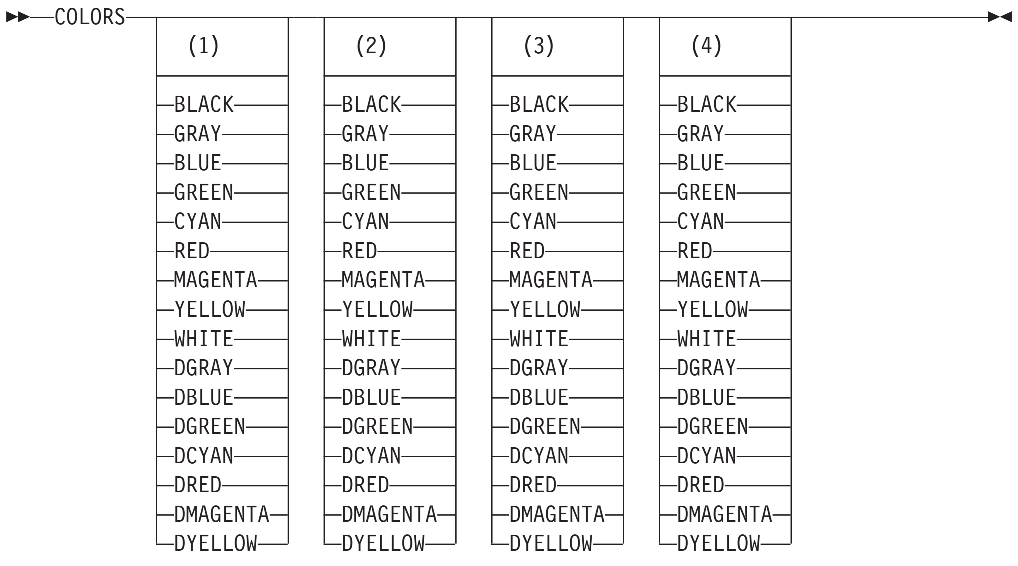


This statement defines an optional title for the pop-up menu. If no TITLE statement appears in the menu file, there is no title on the pop-up menu (the window will have a blank title bar). The title text will be centered at the top of the menu. The final closing quote character is optional unless the title itself contains double quote characters.

Example:

```
title "Command Options
```

COLORS



Notes:

1. Foreground menu color.
2. Background menu color.
3. Foreground bounce bar color.
4. Background bounce bar color.

This statement defines the colors to be used for the pop-up menu. If no COLORS statement appears in the menu file, the CM Mouse default menu colors are used as specified on the CM Mouse control panel. The first color is the menu foreground color; the second is the menu background color; the third is the foreground bounce bar color; the fourth color is the bounce bar background color. The menu colors can be overridden for individual lines of the menu on the LINE statement (see “LINE” on page 48).

Example:

```
colors yellow dblue black blue
```

BAR

```
►►──BAR──┬─────────────────┬──────────────────────────────────────────►◄
         ├─InitialPosition─┤
         └─KEEP────────────┘
```

This statement defines the initial item on which the bounce bar cursor appears. This should be a number between 1 and the number of LINE statements in the menu, or the word “KEEP”. If no BAR statement appears in the menu file, the *InitialPosition* defaults to 1.

Example:

```
bar 3
```

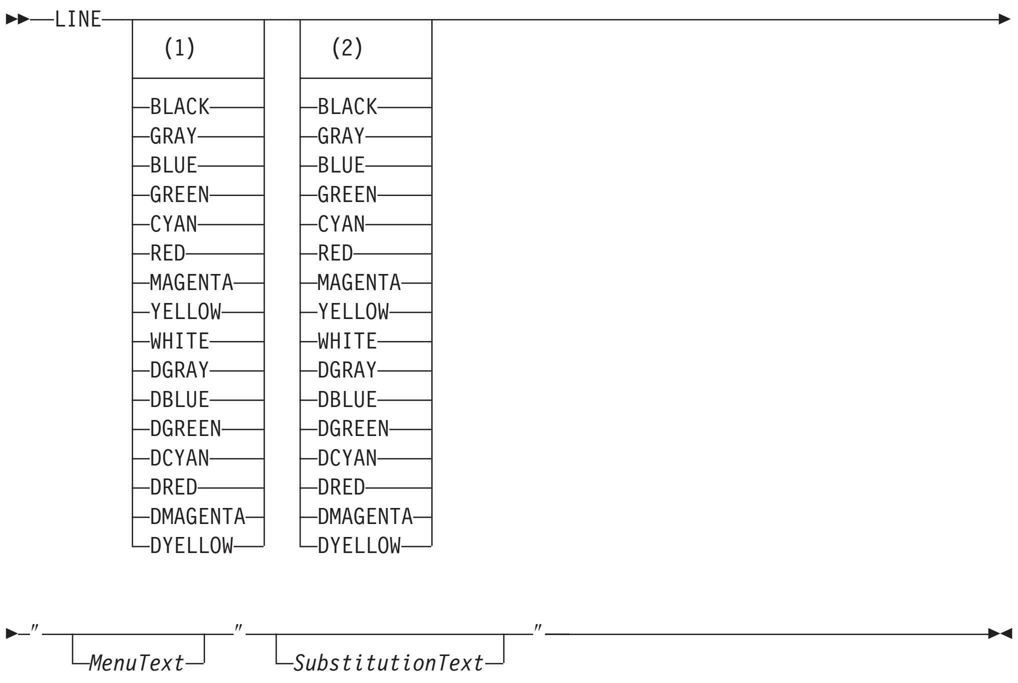
This would cause the bounce bar to be initially positioned on the third item of the menu.

When KEEP is specified, CM Mouse automatically remembers the position of the bounce bar each time the pop-up menu is used. Each time the menu is displayed, the bounce bar is repositioned to the line which was last selected. The line number of the last selected line is saved in a CM Mouse user variable. The name of the variable is the fully qualified name of the menu file which can be obtained with the &{mmm} substitution keyword (see page “&{mmm}” on page 71). The value of this variable can be read and set like any other CM Mouse user variable (see “Chapter 11. CM Mouse Variables” on page 105). For example, consider the menu file:

```
title "Sample"
bar keep
line "Line 1"option a
line "Line 2"option b
line "Line 3"option c{set &{mmm} 2}
line "Line 4"option d
```

This menu will save and restore the bounce bar position each time it is used. However if line 3 is selected, the next time the menu is displayed the bar will be positioned on line 2 rather than 3.

LINE



Notes:

1. Foreground color.
2. Background color.

This statement defines a single selectable item in a pop-up menu list. The two color specifications are optional, and if they are not specified, the menu colors specified on the COLORS statement are used. If no COLORS appears in the menu file, the CM Mouse default menu colors are used. If only one color is specified, it overrides the foreground color, and the default background color is used.

The *MenuText* is the exact text that appears on the pop-up menu. It cannot contain a double quote character. The *SubstitutionText* is any text that is valid in a button definition. If the *MenuText* is selected with the LEFT button when the pop-up menu is displayed, the *SubstitutionText* is substituted into the button definition which invoked this menu.

A blank LINE statement may be used to insert a blank line in the pop-up menu. If a LINE statement does not have to have any *SubstitutionText* the line will be displayed but will not be selectable.

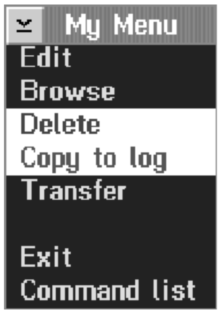
There must be at least one LINE statement in a menu file. The items appear on the menu in the

Like any PM or X/Motif window, the pop-up menu can be moved by using the **Alt-F7** key combination and moving the mouse. This movement technique can be used to expose portions of the menu which normally extend off the screen.

Example:

```
TITLE "My Menu"
LINE "Edit          "{seek}ed[enter]
LINE "Browse        "{seek}browse[enter]
LINE red white   "Delete      "{seek}delete[enter]
LINE green white "Copy to log "{seek}copy * to log[enter]
LINE "Transfer      "{seek}[pf6]
LINE
LINE "Exit          "[pf3]
LINE "Command list  "&{popup cmdlist}
```

This would produce the following menu:
The third line `Delete` would appear in red with a white background, and the next



line would be green on a white background. All other lines would appear in the CM Mouse default menu colors.

If the first item was selected, the host cursor would be moved to the mouse cursor position ({seek}), and the characters `ed` would be typed, followed by the ENTER key. If the last item was selected, the menu `cmdlist` would be displayed. Note that menus are not stacked or nested. In the preceding example, if the last item was selected when the `cmdlist` menu completed (either by selecting an item or the RIGHT button), this menu would not be redisplayed.

SET

```
►►──SET──VariableName──┬─────────┬──"──VariableValue──"──┬─────────┬──────────►◄
                       └─LITERAL─┘                        └─Comment─┘
```

A SET statement can appear anywhere in a menu file. The variable value can be formed from explicit characters and from substitutions. (Note that only *presubstitutions* are allowed in SET statements; no runtime substitutions are allowed.) If the **LITERAL** option is specified then no substitutions are done and the *VariableValue* is used exactly as it appears. SET statements are processed immediately before the menu is displayed. For example:

```
SET CurPos "&{hcol}"
```

This would save the position of the host cursor at the time the pop-up menu was invoked.

See “Chapter 11. CM Mouse Variables” on page 105 for more information on the use of variables.

DOIF/ENDIF

These statements allow lines of the menu excluded based on a specified condition. The excluded lines will not appear when the menu is displayed. The statement format is:

```
►►──DOIF──'string1' ──┬─=───┬──'string2' ──────────────────────────────►◄
                      ├─==──┤
                      ├─!=──┤
                      ├─!==─┤
                      ├─<───┤
                      ├─>───┤
                      ├─<=──┤
                      ├─>=──┤
                      ├─=#──┤
                      └─!=#─┘


►►──ENDIF──────────────────────────────────────────────────────────────►◄
```

Any menu statements between the DOIF and ENDIF statements are skipped if the expression is false. Skipped statements are not processed in any way and are not evaluated for syntax errors. If the expression is true, the statements between the DOIF and ENDIF are read and processed normally.

The comparison operators are the same as those allowed in the {if...} keyword (see page “{if <condition> {then}script1{else}script2}” on page 59). The strings may be formed by any *presubstitution* CM Mouse keywords and/or literal strings. Runtime substitutions are not allowed.

NOTE

Unlike DOIF conditions in BDF script files, there are no limitations on the substitutions which can be used.

The DOIF/ENDIF statements may be nested.

Chapter 8. Button Definitions

A button definition (as defined in a BUTTON statement in the CM Mouse button definition file) can consist of simple characters, special control words, or substitution words. When a button definition contains characters other than control words or substitutions, those characters appear on the host screen when the button is pressed just as if you had typed them on the keyboard.

Characters, substitutions, and control words can be combined to perform a complete function. Control words and characters are processed in the order in which they appear in the definition. Substitutions are either done before the definition is executed (a *presubstitution*) or at execution time (a *runtime substitution*) depending on the position of the ampersand (&) symbol. There are some restrictions on where substitutions can be done (see "CM Mouse Substitution Words" on page 70).

Control and Substitution Word Table

The control words and substitutions which may appear in a button definition are shown in the following tables. Note that some control words are enclosed in square brackets [ ] and some in curly braces { }. The keywords in in brackets represent 3270 keys; those in braces are CM Mouse functions.

CM Mouse Control Word Table

Table 2. CM Mouse Keyword Reference Table

Keyword	Description	Runtime	Environments
{beep}	Sound short beep	-	All
{clip cut} {clip cut textonly}	CUT marked data from host to clipboard	-	All
{clip copy} {clip copy textonly}	COPY marked data from host to clipboard	-	All
{clip copyappend}	Append marked data to clipboard	-	OS2
{clip from <r1> <c1> <r2> <c2>}	Copy specified area of host screen to clipboard	-	All
{clip paste}	Begin a PASTE clipboard operation	-	All
{clip to <r> <c>}	Copy clipboard text to specified area of host screen	-	All
{clip place}	Place a pending clipboard paste operation	-	OS2
{clip cancel}	Cancel a pending clipboard paste operation	-	OS2
{clip clear}	Clear data in the marking box	-	All
{clip undo}	Undo a clipboard paste operation	-	All
{clip unmark}	Remove marking box	-	WIN

Table 2. CM Mouse Keyword Reference Table (continued)

Keyword	**Description**	**Runtime**	**Environments**
{dde <function> <parameters>}	DDE Commands	-	OS/2
{editmmm}	Copy menu file and edit it	-	All
{hostwait n}	Wait *xxx* seconds for keyboard unlock	-	All
{if <condition> {then}script1{else}script2}	Conditional script execution	-	All
{lock on} {lock off}	Lock or unlock the keyboard	-	All
{map}	Display screen map for current session	-	OS2
{mmenu}	Show CM Mouse control panel	-	All
{mrowcol x y}	Move mouse cursor to row *x*, column *y*	-	All
{null}	Do nothing	-	All
{pause n}	Pause (wait) for *n* milliseconds	-	All
{pfkey}	Send PFkey/ENTER left of mouse cursor	-	All
{pfkey first}	Send first PFkey/ENTER on the line	-	All
{pfkey last}	Send last PFkey/ENTER on the line	-	All
{pfkeyrev} {pfkeyrev first} {pfkeyrev last}	Reverse format of {pfkey}	-	All
{pfkeyrev} {pfkeyrev first} {pfkeyrev last}	Reverse format of {pfkey first}	-	All
{pfkeyrev} {pfkeyrev first} {pfkeyrev last}	Reverse format of {pfkey last}	-	All
{printscreen LPTx}	Print the host screen a PC printer	-	All
{printscreen Fname} {printscreen Fname APPEND}	Copy or append the host screen to a PC file	-	All
{quit}	Terminate script execution	-	All
{rowcol x y}	Move host cursor to row *x*, column *y*	-	All
{seek}	Move host cursor to mouse cursor position	-	All
{unseek}	Move mouse cursor to host cursor position	-	All
{search FOR ’string’ AT r1 c1 r2 c2 WAIT n NOT ASIS NOQUIT}	Search host screen for a string	-	All
{seekelse}	Move host cursor, optionally abort	-	All

Table 2. CM Mouse Keyword Reference Table (continued)

Keyword	**Description**	**Runtime**	**Environments**
{seekcol x}	Move host cursor to mouse row, column *x*	-	All
{seekrow x}	Move host cursor to mouse column, row *x*	-	All
{set Name Value}	Set the value of a CM Mouse variable	-	All
{switchto <session>|*|next|prev }	Switch to another host session	-	All
{sys <cmd> <parms>}	Run a system command	-	All
{sysmenu}	Pulldown emulator system menu	-	All
{win <session>|*|prev|next MIN| MAX| RESTORE| HIDE| SHOW| ACTIVATE| DEACTIVATE}	Manipulate specified emulator window	-	All
{xfer ..}	Transfer file	-	All
{?<text>}	Pop-up *<text>* to confirm or abort	-	All
&{break}	Substitute a line break character (hex '0d')	-	All
&{chars r c l} or {&chars r c l}	Chars at row=*r*, col=*c* for length=*l*	✓	All
&{editor}	Name of editor (editor=)	-	All
&{env VarName}	Value of environment variable *VarName*	-	All
&{hcol} or {&hcol}	Current col of host cursor	✓	All
&{hrow} or {&hrow}	Current row of host cursor	✓	All
&{hour} &{min} &{sec}	Current time values (24-hour clock)	-	All
&{month} &{day} &{year}	Current date values	-	All
&{math 'val1' +|-|/|* 'val2'}	Perform mathematical operation	-	All
&{mmm}	Name of current pop-up menu file	-	All
&{mcol} or {&mcol}	Current col of mouse cursor	✓	All
&{mrow} or {&mrow}	Current row of mouse cursor	✓	All
&{num} or {&num}	Numeric string nearest to mouse cursor	✓	All
&{num first} or {&num first}	First numeric string on the line	✓	All
&{num last} or {&num last}	Last numeric string on the line	✓	All
&{num at <row> <col>} or {&num at <row> <col>}	Numeric string nearest row/column specified	✓	All

Table 2. CM Mouse Keyword Reference Table (continued)

Keyword	**Description**	**Runtime**	**Environments**
&{popup <menuname>} or {&popup <menuname>}	Display a pop-up menu, substitute selected line	✓	All
&{sid} or {&sid}	Current host session id (A,B,C...)	-	All
&{rows}	Rows in host screen	-	All
&{cols}	Columns in host screen	-	All
&{srow} &{scol} or {&srow} {&scol}	Row of last successful {search...}	✓	All
&{srow} &{scol} or {&srow} {&scol}	Column of last successful {search...}	✓	All
&{str <function> <parms>}	Perform string functions	✓	All
&{var Name} or {&var Name}	Value of a CM Mouse variable	✓	All
&{word delimit|include '<chars>'} or {&word}	Alphanumeric word nearest to mouse cursor	✓	All
&{word first} or {&word first}	First word on the line of the mouse cursor	✓	All
&{word last} or {&word last}	Last word on the line of the mouse cursor	✓	All
&{word at <row> <col>} or {&word at <row> <col>}	Word nearest to the row and column specified	✓	All
&{?<Qtext>|<Atext>} or {&?<Qtext>|<Atext>}	Prompt user for input using dialog	✓	All
&{rexx PgmSource} or {&rexx PgmSource}	Run a REXX program and substitute EXIT value	✓	OS2
[enter]	3270/5250 ENTER key	-	All
[pf1]-[pf24] [pf01]-[pf09]	3270/5250 Program Function keys	-	All
[newline] [tab] [backtab]	3270/5250 cursor movement keys	-	All
[home] [up] [down] [right] [left]	3270/5250 cursor movement keys	-	All
[fastright] [fastleft]	3270/5250 cursor movement keys	-	All
[pa1] [pa2] [pa3] [attn]	3270 Program Attention keys	-	All
[eraseeof] [delete] [insert]	Misc 3270 keys	-	All
[backspace] [clear] [crsel]	Misc 3270 keys	-	All
[reset] [sysreq]	Misc 3270 keys	-	All
[test]	3270 TEST key	-	All
[fieldmark] [altcsr]	Misc 3270 keys	-	All
[erinp]	Misc 3270 keys	-	All
[fldext] [hex]	Misc 5250 keys	-	OS2
[worddel]	Delete word at host cursor	-	All

Table 2. CM Mouse Keyword Reference Table (continued)

Keyword	Description	Runtime	Environments
[wordright] [wordleft]	Move host cursor 1 word right/left	-	All
[eof]	Move host cursor to end of field	-	All
[pageup] [pagedn]	5250 scroll keys	-	OS2
[help]	5250 help key	-	OS2

The Basics of a Button Definition

A button definition string can contain any text, CM Mouse control words, and CM Mouse substitutions. All of these may be freely mixed in any order. Except for performing substitutions, CM Mouse evaluates ("executes") button definitions from left to right. Thus, if a button definition contains two CM Mouse control words, the left one is executed, and then the right one. Where plain text appears in a button definition, it is sent to the host as simple keystrokes.

For example, consider the button definition:

```
[backtab]s[enter]
```

When this definition is executed by CM Mouse it will cause the host cursor to be tabbed left ("[backtab]" control word), the character "s" to be typed (plain text), and finally the ENTER key to be pressed ("[enter]" control word).

When a button definition contains substitution keywords (those that have an ampersand "&" inside or outside curly braces) the substitutions will usually be done before the button definition is executed. For example:

```
[pf8]Mouse is in row &{mrow}
```

When CM Mouse begins executing this button definition, the first thing it will do is substitute the &{mrow} keyword for the row number of the mouse cursor. So if the mouse cursor was in row 5 at the time this button definition was executed, after substitution the definition would be:

```
[pf8]Mouse is in row 5
```

Since there are no more substitutions in this button definition, CM Mouse begins executing it from left to right. A PF8 keystroke is sent to the host, and then the characters "Mouse is in row 5" are typed on the host screen. Substitutions are discussed in more detail in "Presubstitutions and Runtime Substitutions" on page 80.

All the available CM Mouse control and substitution words are described in the following sections. "Chapter 14. Sample Button Definitions and Menus" on page 117 shows a number of sample button definitions that use these control words.

CM Mouse Control Words

These control words are special commands to CM Mouse to perform some actions specifically related to the mouse. Note that they are enclosed in curly braces “{}”. Any CM Mouse control word can contain presubstitutions or runtime substitutions as described in “CM Mouse Substitution Words” on page 70.

Keywords

The following CM Mouse control words can be used in a button definition:

{beep}

Causes CM Mouse to emit a short beep. It could be used to notify the user of a “dead” (no function) button or of a “dangerous” operation such as deleting a file.

{clip cut} {clip cut textonly}

Removes any currently marked text from the host session and places it in the system clipboard. See “Clipboard Functions” on page 19 for instructions on how to mark an area of text to be cut or copied to the system clipboard. Executing this control word is exactly the same as pressing the **Shift-Delete** key combination. This keyword is not supported on 5250 host sessions.

In OS/2 several views of the host data are stored in the clipboard: for example, a plain text view and a graphic (bit map) view. Using the {clip cut textonly} format of this keyword causes all except the plain text to be removed from the clipboard. This can be useful when pasting to some applications which take the bit map view when you want them to paste the text. Removing the bit map view will force such applications to use the text view.

{clip copy} {clip copy textonly}

Copies any currently marked text to the OS/2 system clipboard. See “Clipboard Functions” on page 19 for instructions on how to mark an area of text to be cut or copied to the system clipboard.

In OS/2 several views of the host data are stored in the clipboard: for example, a plain text view and a graphic (bit map) view. Using the {clip copy textonly} format of this keyword causes all except the plain text to be removed from the clipboard. This can be useful when pasting to some applications which take the bit map view when you want them to paste the text. Removing the bit map view will force such applications to use the text view.

{clip copyappend}

OS/2

Appends any currently marked text to the OS/2 system clipboard. See “Clipboard Functions” on page 19 for instructions on how to mark an area of text to be cut or copied to the system clipboard. Not all emulators support this function.

{clip from <r1> <c1> <r2> <c2>}

Copies a specified rectangular area of the host screen to the system clipboard. *<r1>* and *<c1>* defines the row and column number of one corner of the rectangular area and *<r2>*,*<c2>* defines the other corner. Any of the row/column values can be zero or negative to define an offset from the end of the screen.

The text within the rectangular area will be copied to the system clipboard exactly as it appears on the host screen including protected fields.

Note that this is done without user intervention (the user does not define the clip area with a marking rectangle). Using this keyword with {clip to...}, a script can automate the copy and paste of text to and from the clipboard without any user intervention.

{clip to <r> <c>}

Copies text from the system clipboard to the host screen, starting at the row *<r>* and column *<c>*. The row/column values can be zero or negative to define an offset from the end of the screen.

Each line of text in the system clipboard will be typed on the host screen exactly as it appears in the clipboard. Multiple lines of text are typed on subsequent lines of the host screen, starting in column *<c>*. If the end of the screen is reached, any remaining data in the clipboard is ignored.

Note that the clipboard text is placed on the host screen exactly as if typed manually from the keyboard. If an attempt is made to type over a protected field the emulator may lock the keyboard or issue warning beeps (different emulators will handle this in different ways).

Note that this is done without user intervention (the user does not define the paste area with a marking rectangle). Using this keyword with {clip from...}, a script can automate the copy and paste of text to and from the clipboard without any user intervention.

{clip paste}

Copies any text currently in the system clipboard to the host session. A rectangle appears which indicates the size of the block of text to be pasted. Place the mouse cursor over the rectangle, press and hold the LEFT button, and drag the rectangle to the position desired and release the button. This keyword is not supported on 5250 host sessions.

{clip place}

OS/2

Places text from the clipboard into the host session when the paste rectangle is showing. If a paste operation is not in progress (and no paste rectangle is showing), this keyword has the same effect as sending an ENTER key to the host. This keyword is not supported on 5250 host sessions.

{clip cancel}

OS/2

Cancels a pending clipboard paste operation. This removes a paste rectangle from the screen without affecting the host session. This does not affect the contents of the clipboard. This keyword is not supported on 5250 host sessions.

{clip clear}

Clears the currently marked area of the host screen.

{clip undo}

Reverses the effect of the last clipboard PASTE operation.

{clip unmark}

WIN

Removes the marking rectangle from the screen.

{dde <function> <parameters>}

OS/2

Allows execution of commands in other applications which support DDE. CM Mouse is a DDE client only (it can run DDE commands in other applications, but other applications cannot run CM Mouse commands).

The *<function>* values are:

```
connect 'Name' 'Service'
execute <command>
disconnect
```

Connect is used to establish a connection with a DDE server application. The application and service names must be enclosed in single quote characters. For many applications these values are case sensitive and must be entered exactly as given in the application documentation. A connection must be established before any other DDE function can be used. If the connection fails the remainder of the script will be discarded.

Execute is used to send a command string to a DDE server application. See the application documentation for proper format of this command string.

Disconnect is used to terminate a DDE connection with a DDE server. Many applications support only a single DDE connection at a time so it is important to disconnect from the application after executing DDE commands.

The following example will connect to the OS/2 Excel spreadsheet and instruct the spreadsheet application to select a set of cells, clear them, paste the current contents of the clipboard, generate a graph, and finally activate the spreadsheet window:

```
BUTTON LEFT "{dde connect 'Excel' 'System'}+
  {dde execute +
    [FORMULA.GOTO("R1C2")][SELECT.END(4)][SELECT("R1C2:R[0]C[0]")][CLEAR
(1)]+
    [FORMULA.GOTO("R1C2")][PASTE()]+
    [SELECT.END(4)][SELECT("R1C1:R[0]C[0]")]+
    [NEW(2)][GALLERY.PIE(5)][FULL(TRUE)]+
    [APP.ACTIVATE()]}+
  {dde disconnect}"
```

{editmmm}

Copies the file containing the current pop-up menu definition to the first directory in the MMMDIR path list. When the copy is complete, the text editor is invoked on the copied file. If the file already exists in the first directory of the path, the copy step is skipped. This keyword is identical to selecting the **Edit this menu** option on the pop-up menu system icon (see “Pop-up System Menus” on page 13).

If there is no current pop-up menu (this keyword is in a BDF file), then the most recently displayed pop-up menu is used.

{hostwait n}

Causes CM Mouse to wait until the host has unlocked the keyboard before sending the rest of the button definition. This can be used to prevent host overrun when CM Mouse keystrokes are sent to the host while the host is still processing a previous key sequence. This control word can be abbreviated {hw}. (See “Synchronizing Input with the Host” on page 135 for more information on possible overrun conditions.) If the keyboard is not unlocked within *n* seconds, processing continues anyway.

For example:

```
cmd1[enter]{hw 10}[home][tab]cmd2[enter]
```

In this example, `cmd1` would be entered, and CM Mouse would wait until the host completed the command and the keyboard was ready for input, or for 10 seconds, whichever occurred first. Then the host cursor would be moved, and `cmd2` would be entered.

Note

The {hostwait} keyword is rarely needed since the host emulator queues keystrokes when the keyboard is locked. Keystrokes sent to the host before a previous transaction has completed are sent as soon as the host completes its processing and unlocks the keyboard. Thus, this keyword does not normally need to be used to synchronize input with the host.

{if <condition> {then}script1{else}script2}

Allows conditional execution of scripts. If the *<condition>* is true then *script1* is executed. Otherwise *script2* is executed. The {then} keyword is required, {else} is optional.

The *<condition>* is specified as a comparison of two strings. The full syntax of this keyword is:

```
►►──{if──'str1' ──┬─=───┬──────────────────────────────────────────────────►
                  ├─==──┤
                  ├─!=──┤
                  ├─!==─┤
                  ├─<───┤
                  ├─>───┤
                  ├─<=──┤
                  ├─>=──┤
                  ├─=#──┤
                  └─!=#─┘

►─'str2' ──{then}──┬─────────┬──┬────────┬──┬─────────┬──}──────────────────►◄
                   └─script1─┘  └─{else}─┘  └─script2─┘
```

Note the following syntax rules for this keyword:

1. The string delimiters can be any character that does not appear in the strings themselves.
2. Both strings must use the same delimiter character.
3. Both strings must have delimiters even if they are numeric values.
4. Note that blanks that appear anywhere after the {then} keyword are considered significant and will be part of the script.
5. The {else} keyword and *script2* are optional. The {then} keyword is required.

The comparison operators are as follows:

Operator	Description
=	Case insensitive comparison (upper and lower case are considered equal; the strings ″Abcd″ and ″ABCD″ are equal).
==	Case sensitive comparison (upper and lower case letters are not equivalent; the strings ″Abcd″ and ″ABCD″ are not equal)
!=	Case insensitive not-equal comparison
!==	Case sensitive not-equal comparison
<	Numeric less-than comparison. Both strings must be numeric integer values. Any non-numeric string (such as ″A1″) will be considered zero.
>	Numeric greater-than comparison. Both strings must be numeric integer values. Any non-numeric string (such as ″A1″) will be considered zero.
<=	Numeric less-than or equal.
>=	Numeric greater-then or equal.
=#	Numeric equal. Note that this is different from the string equality (″=″) operator in that the strings ″015″ and ″15″ are numerically equal.
!=#	Numeric not-equal.

If/then/else conditional keywords can be nested to any depth. Note that the closing brace for the keyword appears after the {else} script. For example, the following script would set the variable *X* to different values depending on the position of the mouse pointer:

```
{if '&{mcol}' > '60' +
  {then}{set X Huge}+
  {else}{if '&{mcol}' > '40'+
  {then}{set X Big}+
  {else}{if '&{mcol}' > '20'+
    {then}{set X Medium}+
    {else}{if '&{mcol}' > '10'+
    {then}{set X Small}+
    {else}{set X Tiny}+
  }}}}
```

The following example will look for a blank character on the last line of the screen, fourth character from the right edge. If that character is a blank then a message is displayed, otherwise a command is run on the host system:

```
{if '&{chars 0 -3 1}' = ' '+
   {then}{?You have no mail.}+
   {else}openmail[enter]+
}
```

{lock on} {lock off}

Locks or unlocks the keyboard. If no parameter is specified, ON is assumed. When the keyboard is locked using this keyword, user keystrokes are accepted but deferred until the keyboard is unlocked. The [reset] keyword may be used to discard the stored keystrokes before the keyboard is unlocked.

This keyword can be useful when a long-running button definition is executed, and the user should be prevented from sending keystrokes to the host until the button definition is complete. Using the [reset] keyword before unlocking the keyboard discards any keystrokes the user may have typed during the locked period. For example:

```
{lock on}[pf6]{search for 'Menu 2'
wait 20}run[enter][reset]{lock
off}
```

This button definition will lock the keyboard and then send the PF6 key to the host and wait for `Menu 2` to appear. When it appears, the command `RUN` is typed and the ENTER key is sent. Any keystrokes entered by the user are then discarded and the keyboard is unlocked. This technique can be useful in preventing the user from interfering with long-running automated host interactions.

If the keyboard is not unlocked by a button definition which locks it, CM Mouse will automatically unlock the keyboard after the last keyword of the button definition is completed.

{mmenu}

Restores the CM Mouse control panel. This is equivalent to clicking on the CM Mouse icon and picking the *Restore* option. See “The CM Mouse Control Panel” on page 15.

{mrowcol x y}

Moves the mouse cursor to row *x* and column *y*. Rows and columns are numbered from 1, starting in the upper left corner of the screen. Either value may be specified as zero or a negative value to indicate an offset from the end of the screen. Zero indicates the last row or column, -1 indicates the second from the last row or column, and so on. For example, `{mrowcol -1 0}` would position the mouse cursor in the last column of the screen, one row up from the bottom. Zero and negative values can be used to make positioning independent of the screen size. For example, to position the cursor on the last line of the screen, `{mrowcol 0  1}` could be used and it would work on any size host screen (24 lines, 32 lines, and so on).

{null}

Has no effect and is ignored. It can be used to define an inactive mouse button.

{pause n}

Causes CM Mouse to stop execution for *n* milliseconds. The value *n* should be an integer in the range of 0 to 65536. This can be used to insert short delays into a long command string. This may be necessary under some “host overrun” situations that can cause the host to miss some keystrokes . For example, *{pause 500}* would cause CM Mouse to wait for 1/2 second.

{pfkey}

Sends a PF key or the ENTER key to the host. The particular key is determined by the text on the screen where the mouse is located. An attempt is made to locate a PF or ENTER key definition on the host screen somewhere nearby the mouse cursor. A series of rules is applied in an attempt to intelligently recognize typical host PF key description fields. The search goes as follows:

1. If the cursor is touching a number, it is taken to be the PF key number.
2. If the cursor is on an “F” or a “PF” (capital letters only), the next four characters are searched for a numeric sequence. If found, it is taken to be the PF key number.
3. If the cursor is touching the word “ENTER” (upper case only), the ENTER key is sent. The word “ENTER” must appear in capital letters and must be separated on both sides by non-alphanumeric characters (for example, ENTER would not be recognized in the string “ENTERING”).
4. Starting from the mouse cursor, a search is made to the left looking for a numeric value or the word “ENTER.” If either is found before the left edge of the screen, it is taken to be the PF key number (if numeric) or the ENTER key (if “ENTER”).
5. If none of the preceding locates a potential candidate, or if a numeric value found is out of the range of 1 to 24, a short beep sounds.

For example, the following PF keys would be found:

```
Screen text --->    PF-11    F-11    PF6     F 12    F=6   10    ENTER
Mouse cursor -->     x        x       x      x  x     x     x      x
PF key found -->     11       11      6      6  12    6     10    ENTER


Screen text --->    F6=List    F7=Clear all  PF8 Reset 9: List29
Mouse cursor -->    x     x    x          x  x                 x
PF key found -->    beep  6    7          7  8              beep
```

{pfkey first}

Uses the same algorithm as {pfkey}, except that the first non-blank character of the line is the starting location of the search. This can be useful on some OfficeVision-like menus that list one PF key per line but may contain other characters on the line which can be misleading to the normal {pfkey} control. For example, if a host screen contained the following line:

```
PF4  Show calendar for the last 7 days
```

and the mouse was positioned to the right of the “7,” {pfkey} would recognize PF7, whereas the desired key is actually PF4. Using {pfkey first} causes CM Mouse to start searching at the first non-blank character of the line, and thus it would successfully find PF4 in this case.

{pfkey last}

Uses the same algorithm as {pfkey first}, except that the last character of the line is the starting location of the search.

{pfkeyrev} {pfkeyrev first} {pfkeyrev last}

Same function as the {pfkey...} keywords, except that the format of the PF keys is reversed. Normally CM Mouse expects the PF key descriptions to be of the form:

```
PFx=Description
```

These “reverse format” keywords can be used when the host application displays the descriptions on the left and the PF key number on the right:

```
Description=PFx
```

{printscreen LPTx}

Copies the current contents of the host screen to a system printer. If the LPTx parameter is not specified, LPT1 will be used. Otherwise, “x” should be a single numeric digit indicating which system printer is to be used.

{printscreen Fname} {printscreen Fname APPEND}

Copies the current contents of the host screen to the file name specified. If APPEND is specified, the screen image is appended to the end of the file, otherwise the contents of the file are replaced.

{quit}

Terminates execution of the current script. This can be useful in an {if...} keyword to end the script when a particular condition is met. For example the following script will beep if the number under the mouse cursor is less than 1 or greater than 31. If it is in the range, some keystrokes are sent to the host and the script ends without beeping:

```
{if '&{word}'>'0'+
   {then}{if '&{word}'<'32'+
            {then}curcal[enter]{quit}+
         }+
}
{beep}
```

{rowcol x y}

Moves the host cursor to row *x* and column *y*. Rows and columns are numbered from 1, starting in the upper left corner of the screen. Either value may be specified as zero or a negative value to indicate an offset from the end of the screen. Zero indicates the last row or column, -1 indicates the second from the last row or column, and so on. For example, `{rowcol -1 0}` would position the host cursor in the last column of the screen, one row up from the bottom. Zero and negative values can be used to make positioning independent of the screen size. For example, to position the cursor on the last line of the screen, `{rowcol 0  1}` could be used and it would work on any size host screen (24 lines, 32 lines, and so on).

{seek}

Moves the host cursor to the mouse cursor position (the host cursor will seek the mouse cursor). This is commonly used when the action to be taken depends on the host cursor position (such as when you are selecting from a list). See “Chapter 14. Sample Button Definitions and Menus” on page 117 for examples of how this can be used.

{unseek}

Moves the mouse cursor to the host cursor position which is the opposite of the {seek} control word.

{search FOR ’string’ AT r1 c1 r2 c2 WAIT n NOT ASIS NOQUIT}

This searches the host screen for a specified character string. If the search fails, the remainder of the button definition (to the right of this keyword) is discarded. If the search is successful the remainder of the button definition is executed normally. A successful search will also update the values of the &{srow} and &{scol} substitution variables with the row/column of the located string as well as the SYSTEM_SROW and SYSTEM_SCOL variables.

This keyword has a number of parameters to control the search process. The only required keyword is *FOR*. All other keywords are optional and have default values as explained below. The parameters are:

FOR ’string’

This parameter is required and defines the character string to be used in the search. By default, the string is **not** case sensitive (a search for ″Abcd″ will find the string ″ABCD″).

The ASIS parameter can be used to make the search case sensitive.

The string must be delimited with any single character which does not appear in the string itself. Blanks and tabs may not be used as delimiter characters. For example, the following are equivalent:

```
{search for 'Error 18C'}
{search for XError 18CX}
{search for /Error 18C/}
```

AT row1 col1 row2 col2

This parameter can be used to restrict the search to a specified rectangular area of the screen. If this parameter is not specified, the entire screen is searched. The first row/column values specify the upper-left corner of the search area, and the second row/column values specify the lower-right corner. Any of the row/column values may specify zero or a negative value to indicate an offset from the end of the screen. For example:

```
{search for 'abc' at 1 1 0 0}      Searches entire screen
{search for 'abc' at 1 1 2 0}      Searches first 2 lines
{search for 'abc' at -1 1 0 0}     Searches last 2 lines
{search for 'abc' at 0 40 0 60}    Searches columns 40 to 60 of last line
{search for 'abc' at 5 70 9 80}    Search columns 70-80 in rows 5-9
```

WAIT n

This parameter specifies a timeout value (in seconds). If specified, CM Mouse will continuously search the host screen according to the other search parameters. If the search is not successful within the timeout period then the search fails. The default timeout value is zero, which causes CM

Mouse to perform the search only once (for example, if the string is not found immediately, CM Mouse will not wait for the search string to appear).

VMCLEAR
If a VM ’HOLDING’ or ’MORE...’ state occurs on the host system while searching for a string, a CLEAR keystroke is automatically sent to the host to clear the hold condition. The holding condition is cleared only if a non-zero timeout value has been specified.

NOT This parameter reverses the logical result of the search. If NOT is specified, then a successful search will result in the remainder of the button definition being discarded. If *NOT* is specified and the search fails, then the remainder of the button definition is executed normally.

ASIS This parameter causes the search to be *case sensitive*. When this parameter is used, the FOR string must be specified exactly as it is to be found on the host screen, including upper and lower case characters.

NOQUIT
This parameter will cause the remainder of the script to be executed even if the search fails.

The search parameters may be specified in any order. Thus the following are all equivalent:

```
{search for 'abc' at 1 1 2 0 wait 10 not}
{search not at 1 1 2 0 wait 10 for 'abc'}
{search not wait 10 for 'abc' at 1 1 2 0}
```

{seekelse}

Moves the host cursor to the mouse cursor position and discards the remainder of the definition. If the host cursor and mouse cursor are already at the same location, the remainder of the definition is executed as usual.

{seekcol x}

Moves the host cursor to the same row as the mouse cursor, but in column *x*. Columns are numbered from 1, starting at the left side of the screen. A negative or zero value can be specified to indicate an offset from the right side of the screen. Zero indicates the rightmost column, -1 the next column to the left, and so on. Using zero or a negative value can make the positioning independent of the screen width. For example, `{seekcol 0}` positions the cursor in the rightmost column on any size host screen (80 columns, 132 columns and so on). This control word can be very useful for moving the host cursor to a known position within a multicolumn list.

For example:

```
{seekcol 45}[crsel]
```

This would move the host cursor to the same line as the mouse cursor, in column 45. A light-pen selection would then be sent to the host.

{seekrow x}

Moves the host cursor to the same column as the mouse cursor, but in row *x*. Rows are numbered from 1, starting at the top of the screen. A negative or zero value can be specified to indicate an offset from the bottom of the screen. Zero indicates the last line, -1 the next to last line, and so on. Using zero or a negative value can make the positioning independent of the screen size. For example, `{seekrow 0}` positions the cursor in the last row on any size host screen (24 rows, 32 rows and so on).

{set Name Value}

Sets the value of a user-defined variable. The *Name* can be any sequence of characters, excluding blanks. The *Value* is any string and can be formed by substitutions. There must be exactly one blank between the name and the first character of the value. Extra blanks will be considered part of the value. For example:

```
    {set MyOwnVariable The host cursor is at column &{hcol}}
```

This sets the variable *MyOwnVariable* to the string "The host cursor is at column x", where *x* is the column number of the host cursor position.

See "Chapter 11. CM Mouse Variables" on page 105 for more examples and a complete discussion of user-defined variables.

{switchto <session>|*|next|prev }

Connects to the specified session. The *<session>* specified must be the one-character short session ID (A, B, C, etc.) or one of the special values:

- ***:** (asterisk) This value represents the current session to which CM Mouse is connected.
- **next:** This value represents the next session after the current session. Sessions are ordered alphabetically from A to Z.
- **prev:** This value represents the previous session to the current session. Sessions are ordered alphabetically from A to Z.

After this keyword is executed, all CM Mouse host interactions will be with the new session. Note that this keyword only causes CM Mouse to interact with a new session; it does not activate, maximize, or affect the emulator windows in any way.

{sys <cmd> <parms>}

Causes CM Mouse to run the command *<cmd>* with the parameters *<parms>*. CM Mouse passes the command to the operating system to be run. The way in which the command is run is different in each environment, so each is described separately.

In OS/2 the command is run as a separate asynchronous process. This means that CM Mouse starts the command running but does not wait for it to finish. The remainder of the script will continue executing while the command runs.

CM Mouse does not run the OS/2 command processor to execute the command. Note the following rules for specifying the command (program) to be executed:

- The file name must have the extension specified.
- If the executable file is not in the OS/2 PATH then the fully qualified file name must be supplied.

- If the command is an OS/2 internal command (such as DIR) then the OS/2 command processor must be explicitly executed.
- OS/2 command files (.CMD) must be executed via the the OS/2 command processor (CMD.EXE).

For example, the following would copy a file. Note that the OS/2 command processor is used since COPY is an internal OS/2 command:

```
{sys cmd.exe /C copy c:\config.sys c:\config.sav}
```

In the following example, the OS/2 system editor is executed to edit a CONFIG.SYS file:

```
{sys e.exe c:\config.sys}
```

WIN

In Windows the command is run as a separate Windows process. CM Mouse starts the command running but does not wait for it to finish. The remainder of the script will continue to execute while the command runs.

The command may be a DOS command or a Windows application program. If it is an internal DOS command, the DOS command processor (COMMAND.COM) must be explicitly run. For example:

```
 {sys command.com /K dir c:\}
```

If the command file name is not fully qualified a search will be made for the program in (1) the current directory, (2) the Windows directory, (3) the Windows 'system' directory, and (4) the DOS PATH directories.

{win <session>|*|prev|next MIN| MAX| RESTORE| HIDE| SHOW| ACTIVATE| DEACTIVATE}

Manipulates the specified session window. The *<session>* can be a single character short session ID or one of the following:

- ***:** (asterisk) This value represents the current session to which CM Mouse is connected.
- **next:** This value represents the next session after the current session. Sessions are ordered alphabetically from A to Z.
- **prev:** This value represents the previous session to the current session. Sessions are ordered alphabetically from A to Z.

Note that this does not have to be the same session to which CM Mouse is currently connected. Combinations of window conditions may be specified to achieve the desired effect. For example the following would un-hide the emulator window (if hidden) and activate it (bring it to the top and give it focus). This would not change the minimize/maximize/restore state of the window:

```
{win * show activate}
```

The following would maximize the next available host session:

```
{win next show max activate}
```

{xfer ..}

Transfer a file to or from the host system. The syntax of this keyword is:

```
►►──{──XFER──┬─SEND─┬──TYPE=──┬─VM──┬──PC=──┬─PCFileName─┬──HOST=──────────────►
             ├─RECV─┤         └─MVS─┘       └─*──────────┘
             ├─DRAG─┤
             └─DROP─┘

►──┬─HostFileName─┬──┬──────────────────┬──┬──────────────────────┬───────────►
   └─*────────────┘  └─SESS=──┬─A───┬───┘  └─XLATE=──┬─*─────────┬─┘
                              ├─B───┤                └─XlateName─┘
                              ├─...─┤
                              └─*───┘

►──┬───────────────────────────┬──┬───────────────────┬──┬──────────────────────┬──►
   └─METHOD=──┬─EHLLAPI─┬──────┘  └─QUIET=──┬─YES─┬───┘  └─CLEAR=──┬─YES─────┬───┘
              └─SCREEN──┘                   └─NO──┘                ├─NO──────┤
                                                                   └─DEFAULT─┘

►──┬────────────────────────────┬──}──────────────────────────────────────────►◄
   └─PARMS=──XferParameters─────┘
```

Each of the parameters and options are described in detail in “Chapter 10. Drag/Drop Features” on page 97.

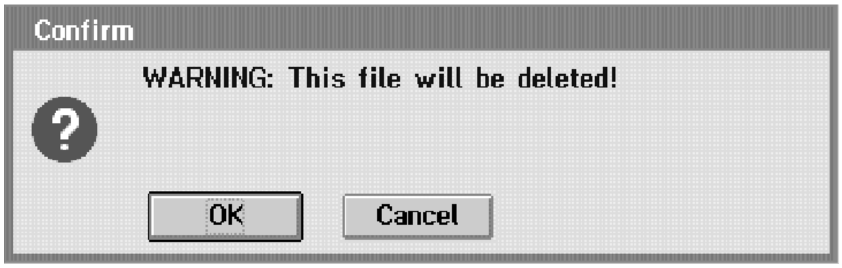
{?<text>}

Causes a pop-up menu to appear containing the specified text with **OK** and **Cancel** buttons. The user can select the **OK** button to confirm the action or the **Cancel** button to abort it. If the user presses the **Cancel** button, the remainder of the button definition (after the {?...} keyword) is discarded. If the **OK** button is pressed, the remainder of the definition is executed normally.

This keyword could be used to confirm a risky action and provide the user with an easy way to abort it. For example, the keyword

```
{?WARNING: This file will be deleted!}delete[enter]
```

would cause the following pop-up menu to display:


Confirm

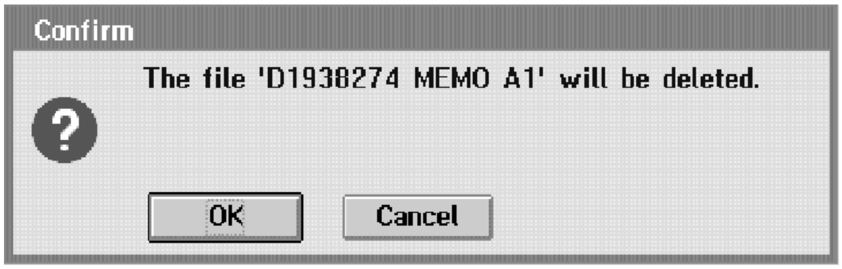
WARNING: This file will be deleted!

If the user pressed the **OK** button, then the characters `delete` would appear on the host screen followed by an ENTER key. If the user pressed the **Cancel** button, the remainder of the definition would be ignored and nothing would happen.

Note that the text can be constructed from CM Mouse substitution keywords (see “CM Mouse Substitution Words” on page 70). For example, the following might be used to confirm the deletion of a file on a VM FILELIST screen:

```
{?The file '&{chars &{mrow} 1 20}' will be deleted.}erase[enter]
```

This would substitute the first 20 characters from the line the mouse cursor was on into the text (see “CM Mouse Substitution Words” on page 70 for information on the &{char} and &{mrow} keywords). This example would produce a pop-up like:
This keyword should not be confused with the substitution word which has the


Confirm

The file 'D1938274 MEMO A1' will be deleted.

same form but contains a substitution symbol (&) inside or outside of the braces.

{map}

OS/2

Displays the Screen Map window for the current host session. See “CM Mouse Screen Map Facility” on page 20.

{sysmenu}

Displays the standard host emulator system menu pull-down. This pull-down is not directly available because CM Mouse intercepts mouse clicks on the emulator’s system menu icon (see “The Host System Menu” on page 18). When this control word is executed, the emulator’s pull-down menu is invoked so that you can perform special emulator operations such as changing font sizes.

CM Mouse Substitution Words

The substitution words described in the following paragraphs can be used in CM Mouse BUTTON statements of BDF files, and LINE and TITLE statements of MMM (menu) files. These substitution words are replaced with a value during execution so that you can construct definitions which depend on runtime values such as the position of the mouse or host cursors. Substitutions can be nested and may occur in CM Mouse control words or host keystrokes.

The CM Mouse substitution words are as follows:

&{break}

This substitution word will substitute a single line-break character. This can be used to insert line breaks in lengthy inline REXX programs. Line breaks are required in REXX programs at least every 512 characters of source code. The break must appear where REXX allows a normal end-of-line. For example:

```
&{rexx +
        /* REXX code */ +
        xyz = 'ABC'; +
        &{break} +
        if x=y +
          then ... +
      }
```

This keyword is useful only in OS/2.

&{chars r c l} or {&chars r c l}

Substitutes a string of characters from the host screen starting at row *r*, column *c*, and of a length *l*. All three values must be specified (or formed from substitutions). For example, the following display on the host screen the five characters that occur at row 2 column 1, followed by an ENTER key:

```
&{chars 2 1 5}[enter]
```

The following takes five characters starting at the current position of the mouse cursor and displays them on the host screen followed by an ENTER key. Note the use of nested substitution words:

```
&{chars &{mrow} &{mcol} 5}[enter]
```

The row and column values can be specified as zero or negative numbers to indicate an offset from the end of the screen. For example, a row number or zero indicates the last line of the screen, -1 is the second from last line of the screen, and so on. This allows the position to be independent of screen size if necessary. For example the following will take the last 5 characters from the last row of any size host screen:

```
&{chars 0 -4 5}
```

&{editor}

Substitutes the name of the PC text editor specified with the “editor=” parameter (or defaulted) when CM Mouse was started.

&{env VarName}

Substitutes the value of the of the environment variable named. If the specified environment variable does not exist, a null string is substituted.

&{hcol} or {&hcol}

Substitutes the column number of the current position of the host cursor.

&{hrow} or {&hrow}

Substitutes the row number of the current position of the host cursor.

&{hour} &{min} &{sec}

Substitutes the current time of day based on the PCs internal clock. The range of values is: hour (00-23), min (00-59), sec (00-59). These keywords always substitute a 2-character value (leading zeros are appended to values less than 10).

&{month} &{day} &{year}

Substitutes the current month (01-12), day (01-31), and year (90) from the PCs internal clock. These keywords always substitute a 2-character value (leading zeros are appended to values less than 10).

&{math ’val1’ +|-|/|* ’val2’}

Substitutes the results of the specified integer mathematical operation. Both values must be enclosed in a delimiter character. The delimiter may be any character which does not appear in the value itself. Both values must use the same delimiter.

The operators are:

```
+       Addition
-       Subtraction
/       Division
*       Multiplication
```

The string values must be numeric. Any non-numeric string is taken as zero. A division by zero produces a value of zero. Values may be any positive or negative integer values (no decimal points) in the range (approximately) -2 billion to +2 billion.

The following example will add one to the number under the mouse cursor:

```
&{math /&{word}/ + /1/}
```

&{mmm}

Substitutes the PC file name of the current pop-up menu. If this substitution word occurs in a BDF file (rather than a pop-up menu), the name of the most recently used pop-up menu file is substituted. The file name includes the complete path name.

&{mrow} or {&mrow}

Substitutes the row number of the current position of the mouse cursor.

&{mcol} or {&mcol}

Substitutes the column number of the current position of the mouse cursor.

&{num} or {&num}

Substitutes the number on the host screen that is nearby the mouse cursor position. "Nearby" is determined by the following rules:

1. If the mouse cursor is currently on a numeric character, all adjoining numeric characters to the left and right are used.
2. A search is made to the left for a numeric character. If found, it is used to find the number as in (1).
3. A search is made to the right for a numeric character. If found, it is used to find the number as in (1).

If no numeric characters are found on the line the result is a null string. "Numerics" consist of the characters 0 through 9.

This substitution keyword can be useful for host applications that present a list of selections and prefix the list by command numbers to be entered. This can even be used on multicolumn lists. For example, a host screen might look like this:

```
Enter Command ===>

  0. Erase file     1. Delete maps      3. Remove all
  4. Copy           5. Rename           6. Send
  7. Edit overlay   8. Undo changes     9. Load new copy
```

By defining a mouse button to be "&{num}[enter]", you can position the mouse cursor on any command in the list and select the option by pressing that mouse button. Note that the &{word} keyword would not be useful here because we are interested only in the command numbers, not the words that make up the command descriptions.

&{num first} or {&num first}

The same as &{num}, except that the search begins at the first character of the line the mouse is on.

&{num last} or {&num last}

The same as &{num} except that the search begins at the last character of the line the mouse is on.

&{num at <row> <col>} or {&num at <row> <col>}

The same as &{num} except that the search begins at the row and column specified. Rows and columns are numbered from 1, starting at the upper left corner of the screen. The row and column values can be specified as zero or negative values to indicate an offset from the end of the screen. Zero indicates the last row (or column), -1 the second to last, -2 the third, and so on. For example, `&{num at -1 0}` would substitute a number from the second to last line of the screen, starting at the rightmost column.

&{popup <menuname>} or {&popup <menuname>}

Causes CM Mouse to display the pop-up menu specified. The *<menuname>* specified is the name of a file in the MMMDIR path (specified when CM Mouse was started). The name may optionally include a *relative* path name that is appended to each directory in the path until the file is found.

The substituted value is the button definition string from the line the user selects on the pop-up menu. Any presubstitutions in that button definition are done during the execution of this keyword.

For example,

```
&{popup picklist}[enter]
```

would cause CM Mouse to read the menu file *<mmmdir>*\PICKLIST.MMM and display the pop-up menu on the host screen.

For example, if the default *mmmdir* parameter was used, the full file name would be C:\CMMOUSE\PICKLIST.MMM. When the user selects a line of that menu, the button definition associated with that line is substituted. The substituted value is then evaluated for any further substitutions and, when all substitutions are done, the definition is executed from left to right. Suppose, for example, that the PICKLIST.MMM file for the preceding example contained:

```
line "Option 1"command(start)"
line "Option 2"command(quit)"
line "Option 3"exit[enter][pf6]
line "Option 4"[home]&{chars 10 1 8}
```

When the preceding button definition is executed, the first thing to be done is to evaluate the &{popup picklist& rbrc. presubstitution. This causes a four-line pop-up menu to appear with options 1 through 4. The user can then pick any one of the four lines.

If, for example, the user picks the first line (“Option 1”), the characters `command(start)` are substituted into the original button definition. The definition would then look like:

```
command(start)[enter]
```

Because there are no more substitutions to do, the definition is executed from left to right, causing the characters `command(start)` to be typed on the host screen, followed by an ENTER key.

Suppose now that the user picked the “Option 4” line of the pop-up menu. The substituted definition would be:

```
[home]&{chars 10 1 8}[enter]
```

The substitution in this definition would then be evaluated, causing the first 8 characters of row 10 on the host screen to be substituted in place of “&{chars 10 1 8}”. The definition would be executed, causing the host cursor to be placed at the first input field, the 8-character string would be typed, and an ENTER key would be sent.

If the user cancels a pop-up menu which results from a substitution, the remainder of the button definition is discarded. In the preceding example, if the user canceled the pop-up menu (by clicking the RIGHT button or selecting “Close” from the pop-up’s system menu), no substitution would be done, and no keystrokes would be sent to the host. Note that in this case it is important to distinguish between a presubstitution pop-up and a runtime substitution pop-up. If a presubstitution pop-up is canceled, the entire button definition is discarded. If a runtime substitution pop-up is canceled, only the definition to the right of the {&popup...} keyword is discarded. For example, consider the following definition:

```
run options=&{popup options}[enter]
```

As written, this definition would first display the OPTIONS.MMM menu. If the user selected a line of that menu, the button definition from that line would be substituted and the characters `run...` would be typed on the host screen followed by an ENTER key. However, if the user canceled the pop-up menu, no keystrokes would be sent to the host (not even the keystrokes to the left of the &{popup options} keyword).

Now consider the slightly different definition:

```
run options={&popup options}[enter]
```

Note that the ampersand (&) is inside the braces, making the pop-up keyword a runtime substitution. When this definition is executed, the first thing that happens is the characters `run options=` are typed on the host screen. *Then* the OPTIONS.MMM pop-up menu is displayed. If the user selects a line of the menu, it is substituted, typed on the host screen, and the ENTER key is sent. However, if the user cancels the pop-up menu, the remainder of the definition is discarded but the characters `run options=` remain on the host screen.

See Chapter 7. CM Mouse Menu Files for information on the format of menu files.

&{rows}

Substitutes the number of rows in the host screen.

&{cols}

Substitutes the number of columns in the host screen.

&{sid} or {&sid}

Substitutes the value of the current host session ID (“A”, “B”...).

&{srow} &{scol} or {&srow} {&scol}

Substitutes the row/column number of the last successful {search} keyword execution.

&{str <function> <parms>}

Substitutes the result of applying the specified function to strings. Strings shown enclosed in quotes in the following table may be enclosed in any delimiter character that does not appear in the string itself. If more than one string is used, they must all use the same delimiter. Numeric values are not enclosed in delimiters.

In general these functions are identical in operation to the OS/2 REXX string functions. The supported functions and their parameters are:

Function	Parameters	Description
substr	’<string>’ start length	Takes a substring of *<string>* starting at the specified position. The *length* is optional and if not specified, the remainder of the string is taken. If the string is not long enough for the *length* specified it will be padded with blanks.
strip	’<string>’ L|T|B <char>	Returns the *<string>* stripped of leading (*L*), trailing (*T*), or both (*B*) characters. The character to be stripped is the blank character unless otherwise specified. Note that the strip character is not enclosed in any delimiters.
word	’<string>’ N <char>	Returns the *N*'th word of *<string>* delimited by the specified character. If no delimiter is specified then blank is used. If the string contains less than *N* words, a null string is returned.
words	’<string>’ <char>	Returns the number of words in *<string>* delimited by the specified character. If no delimiter is specified then blank is used.
right	’<string>’ length	Returns the rightmost characters of *<string>*. If the string is less than the length specified the result is padded on the left with blanks.
length	’<string>’	Returns the number of characters in *<string>*.
pos	’<needle>’ ’<haystack>’	Returns the position of the *<needle>* string in the *<haystack>* string. If the needle does not appear anywhere in the haystack, zero is returned.
concat	’<Lstrng>’ ’<Rstrng>’	Returns the concatenation of the *<Lstring>* (on the left) and the *<Rstring>* (on the right).
insert	’<source>’ ’<target>’ index	Inserts the *<source>* string into the *<target>* string starting at the specified position.
delete	’<string>’ start length	Deletes characters from *<string>* starting at position *start*. If no length is specified, the remainder of the string is deleted.
upper	’<string>’	Converts all characters in *<string>* to upper case.

Examples:

```
Convert to upper case and remove leading & trailing blanks

  &{str upper '&{str strip ' partA of 1  ' B}'}  = "PARTA OF 1"

Take 2nd qualifier of an MVS dataset name:

  &{str word /TMAMB65.BEGIN.CLIST/ 2 .}  = "BEGIN"

Take last qualifier of an MVS dataset name:

  {set dsname TMAMB65.PROJECT.JCL.JOB16}+
  {&str word '{&var dsname}' {&str words '{&var dsname}' .} .}  = "JOB16"
```

&{var Name} or {&var Name}

Substitutes the value of the variable named. Note that if the presubstitution form of the keyword is used, the substitution is done before any part of the button definition is executed (and thus {set...} statements in the definition for this variable will have no effect). See “Chapter 11. CM Mouse Variables” on page 105 for a complete discussion of variables.

&{word delimit|include ’<chars>’} or {&word}

Substitutes the “word” on the host screen that is nearest to the mouse cursor position. The term “nearest” means the following:

1. If the mouse cursor is currently on an alphanumeric character, all adjoining alphanumeric characters to the left and right are used.
2. A search is made in both directions at once, and the first alphanumeric character found is used to find the word as in (1).
3. If the mouse cursor lies exactly between two words, the right word is taken.

If no alphanumeric characters are found on the line the result is a null string. By default (if neither *delimit* or *include* is specified) alphanumerics consist of upper and lower case A-Z, 0-9, and the special characters “@”, “$”, “#”, “-”, and “_”.

If *delimit* is specified, a list of characters is given which are used to delimit the word. All other characters will be included in the word.

If *include* is specified, a list of characters is given which will be included in the word, and all other characters are considered delimiters. The alphanumerics A-Z and 0-9 are always included.

Both *delimit* and *include* cannot be specified.

Examples:

```
In the following examples assume that the mouse pointer is
positioned over the letter "A" in the string

      " _ !Hello$A1 Goodbye!"

&{word}              = "Hello$A1"
&{word delimit ' '}  = "!Hello$A1"
&{word delimit '$!'} = "A1 Goodbye"
&{word include '$!'} = "!Hello$A1"
```

&{word first} or {&word first}

The same as &{word}, except that the search begins at the first character of the line the mouse is on. This can be useful for host applications that present a list of selections and prefix the list by names or numbers to be entered. For example, a typical host screen might look like this:

```
OPTION ===> _

  0  ISPF PARMS
  1  BROWSE
  2  EDIT
  3  UTILITIES
```

By defining a mouse button to be “&{word first}[enter]”, you can position the mouse cursor anywhere on the line and select the option by pressing that mouse button.

&{word last} or {&word last}

The same as &{word} except that the search begins at the last character of the line the mouse is on.

&{word at <row> <col>} or {&word at <row> <col>}

The same as &{word} except that the search begins at the row and column specified. Rows and columns are numbered from 1, starting at the upper left corner of the screen. The row and column values can be specified as zero or negative values to indicate an offset from the end of the screen. Zero indicates the last row (or column), -1 the second to last, -2 the third, and so on. For example, &{word at -1 0} would substitute a word from the second to last line of the screen starting at the rightmost column.

&{?<Qtext>|<Atext>} or {&?<Qtext>|<Atext>}

Displays a dialog panel for the user to type in a string. The dialog panel consists of a prompting string at the top, a text input field the user can modify, and two buttons marked **OK** and **Cancel**. The string the user types in the input field will be the substituted value of the keyword.

Two text strings can be specified in this keyword. The first is the prompt string displayed at the top of the dialog box. The second string is optional and, if present, must be separated from the first string by a vertical bar character (|). The second string is used to preset the input field.

If the second string begins with a percent sign (%), the input field will not show the characters typed by the user. This can be useful for password prompts in which the string should not appear on the screen.

If the second string begins with a dollar sign ($) then the string is taken to be the name of a CM Mouse variable. The value of the named variable will be used to preset the input field. When the user selects **OK**, the string the user typed in the input field is used to update the value of the variable.

If both ‘%’ and ‘$’ are specified the ‘%’ must be first. For example:

```
&{?Enter password:|%$PwValue}
```

The dialog box is positioned such that the upper left corner is one row below and one column to the right of the mouse cursor but fit to the screen edges. If the user clicks on the **OK** button or presses the **ENTER** key, this keyword is replaced by the value of the input field of the dialog. If the user clicks on the **Cancel** button or presses the **Esc** key, nothing is substituted and the remainder of the button definition is discarded.

The user can type up to 255 characters into the text field.

This substitution keyword follows the semantics of presubstitution and runtime-substitution keywords. If used as a presubstitution (the “&” outside the braces), the input dialog is shown to the user *before* any part of the button definition is executed. If the user presses the **Cancel** button, the entire definition is discarded.

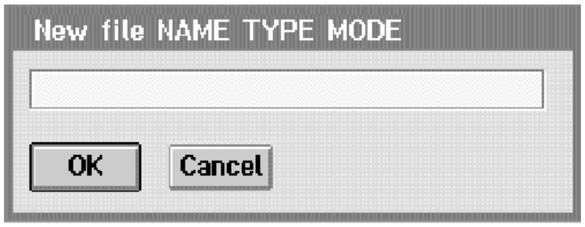
If, however, this keyword is used as a runtime substitution (the "&" inside the braces), then any keywords or host keystrokes to the left of this substitution word are executed and sent to the host *before* the dialog is shown to the user.

Note that any part of the question or answer text of this keyword can be constructed with other substitution keywords.

Example 1:

```
{seekcol 20}rename / &{?New file NAME TYPE MODE}[enter]
```

This example might be used on a FILELIST screen. It would first present a dialog screen similar to the following:
Note that the dialog will be shown before any part of the definition is executed. If



the user types a string and then presses **ENTER**, the host cursor will be repositioned and a RENAME command will be typed and entered.

Example 2:

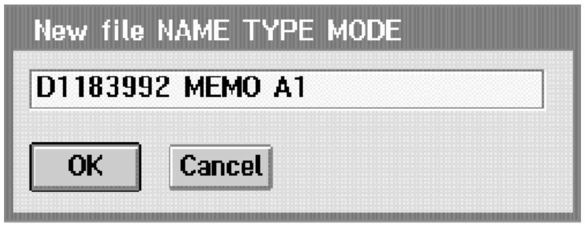
```
{seekcol 20}rename / {&?New file NAME TYPE MODE}[enter]
```

This example is identical to the previous one except that a runtime substitution is being used. In this case, the host cursor is repositioned and a partial RENAME command is typed on the host screen before the dialog is presented. If the user presses **ENTER**, the RENAME command completes. However, if the user cancels the dialog, the partial RENAME command remains on the host screen. In this case, the previous example (with a presubstitution) is preferred since it won't leave a partial command on the host screen if the user chooses to cancel the operation.

Example 3:

```
{seekcol 20}rename / &{?New file FN FT FM|&{chars &{mrow} 1 20}}[enter]
```

In this example, the input field will be filled in for the user with a default value. The value will be the first 20 characters of the line the mouse is on. On a FILELIST screen, that is the file name, type, and mode. The dialog will appear similar to the following:
with the input field already filled in. The user can type over any part of the input

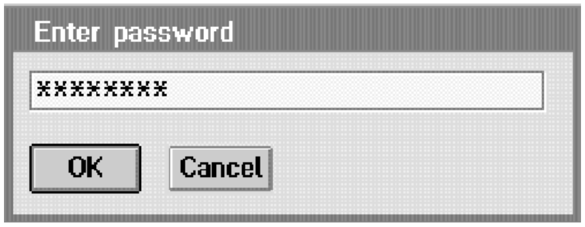


field or press **ENTER** to accept the default value (which for this rename operation would not make much sense).

Example 4:

```
ralvm18 mcmillan{search for 'password' wait 30}&{?Enter password|%}[enter]
```

In this example, an input dialog is first shown like:
As characters are typed, they are not shown on the screen. In this example, if



ENTER is pressed (or **OK** selected), a logon sequence is initiated with the host and CM Mouse waits 30 seconds for the string `password` to appear. If it appears, the value typed on the dialog panel is entered. If `password` fails to appear in 30 seconds, no further action is taken.

&{rexx PgmSource} or {&rexx PgmSource}

OS/2

This substitution word will run a REXX program. The substituted value will be the EXIT value of the REXX program. The source statements for the REXX program are specified as part of the substitution keyword. Normal CM Mouse substitution rules apply to the *PgmSource* part of this keyword. If the source code contains CM Mouse presubstitutions, then they are done before any part of the button definition is executed. If the source contains runtime substitutions, they are done when this keyword is evaluated but before the REXX program is run.

This substitution word allows a complete REXX-language program to be imbedded ("inline") within a CM Mouse button definition. Any series of valid REXX language statements can be used to write a complete REXX program. The syntax of the *PgmSource* part of this keyword must follow normal REXX syntax rules. In addition, CM Mouse line continuation and quoting rules must also be followed. To write a complete inline REXX program, the following rules must be observed:

1. If the REXX program is more than a single line, then normal CM Mouse line continuation characters must be used.
2. There are no implied REXX statement delimiters at the end of lines. Every REXX statement must be explicitly terminated with a semicolon.
3. If the REXX program exceeds 512 characters in length, the &{break} keyword must be used to break the program into sections of 512 characters or less.
4. If the REXX program is to contain double-quote characters, then the entire button definition in which the &{rexx ...} substitution appears must end in a double-quote character.

For example, the following button definition will cause either a PF3 or PF7 to be sent to the host depending on the position of the host cursor. In either case, the **PF** key is followed by an **ENTER** key.

```
button left "&{rexx if &{hrow}=1 +
               then exit '[pf3]'; +
               else exit '[pf7]'; +
            }[enter]"
```

Notice that every line of the button definition is continued with a CM Mouse line continuation character. Also notice that each REXX statement is explicitly terminated with a semicolon. The CM Mouse substitution word &{hrow} is substituted into the REXX source code before the button definition is executed.

For a complete discussion of the &{rexx ...} substitution keyword, see “Chapter 9. CM Mouse/REXX Interface” on page 83.

Presubstitutions and Runtime Substitutions

There are two types of substitutions, *presubstitutions* and *runtime substitutions*. Presubstitution is valid with all CM Mouse substitution keywords, but runtime substitution is valid only for some of them. Presubstitution is indicated by an ampersand (&) outside of the braces, while runtime substitution is indicated by an ampersand inside the braces.

A presubstitution is replaced with its value before any part of the button definition is executed. A runtime substitution is replaced with its value when it is encountered in the button definition as the definition is executed. These two types of substitutions produce different results when something in the button definition changes the value to be substituted before the substitution word occurs. For example, consider the following button definition:

```
[right][right][right]Host cursor is at position &{hcol}
```

If the host cursor was in column 10 when this definition was executed, the following string would be typed on the host screen:

```
Host cursor is at position 10
```

The host cursor position was substituted into the string *before* any part of the button definition was executed, and thus the value “10” was substituted. If the button definition was:

```
[right][right][right]Host cursor is at position {&hcol}
```

(note the ampersand is now inside the braces), the value “13” would be substituted because the [right] keywords would move the host cursor from column 10 to 13, and *then* the {&hcol} would be substituted with the position of the host cursor (which is then 13). Thus, the following would be typed on the screen:

```
Host cursor is at position 13
```

In general, presubstitutions and runtime substitutions can be freely mixed within the same button definition:

```
[right][right][right]Host was at &{hcol}, is now at {&hcol}
```

This would show the position of the host cursor before the definition started and when it was completed. The only time they *cannot* be mixed is in a nested substitution in which the outer substitution is a presubstitution and the inner is a runtime substitution. For example, the following construct is not allowed and will produce an error message when executed:

```
&{chars {&hrow} 1 18}
```

This is invalid because the outer substitution is to be done before anything is executed, but it contains a substitution which must be done at runtime. The following definition is valid:

```
   {&chars &{hrow} 1 18}
```

In this case, the inner substitution `&{hrow}` is done before any part of the definition is executed. The `{&chars}` substitution is then done when it is executed and there is no problem because the `&{hrow}` has already been evaluated and replaced.

As another example of the difference between presubstitution and runtime substitution, consider a button definition which invokes some host function and then reads data (placed there by the function) from the host screen with the `&{chars r c l}` substitution word. For example, we could invoke OfficeVision, get the current time from the OfficeVision main menu, and construct some fictitious command using it.

```
  office[enter]{vmwait}timewarp nextstop=&{chars 3 64 8}
```

The previous sequence would not work as you might expect because the `&{chars...}` is substituted *before any part of the definition is executed*. It is evaluated to be whatever is on the screen *before* the OfficeVision command is even typed on the host screen.

By placing the substitution character inside the braces to make it a runtime substitution, CM Mouse will evaluate the substitution when the keyword is encountered during processing:

```
  office[enter]{vmwait}timewarp nextstop={&chars 3 64 8}
```

This would enter the OFFICE command, wait until the main menu was displayed, type the characters *"timewarp nextstop="* on the host screen, and *then* read the third row, 65th position and get an 8-character string to be typed on the screen.

Both presubstitutions and runtime substitutions are done from left to right. For example, consider the following button definition:

```
   &{?First name} &{?Last name}
```

When executed, this definition will first display an input dialog asking for `First name`. When that dialog is complete, another will be displayed asking for `Last name`. This would behave in the same manner if both of the preceding were runtime substitutions.

CM Mouse Host Control Words

Host control words are instructions to the emulator program to perform some interaction with the host or to control the emulator itself. Host control words are enclosed in square brackets "[ ]".

Most of the host control words correspond directly to a key on the keyboard such as [pf1], [pf2], or [clear]. A control word exists for almost every special key on the keyboard including cursor controls, PF keys, insert and delete keys, tabs, and PA keys. Any of the control words shown in "Control and Substitution Word Table" on page 51 can be used within a CM Mouse button definition.

Chapter 9. CM Mouse/REXX Interface

REXX is a general purpose programming language which is built into the OS/2 operating system. REXX has the usual structured programming instructions IF, THEN, ELSE, DO and so on. It is **not** necessary to know REXX to make effective use of CM Mouse. However, REXX can be used to extend the capabilities of CM Mouse. This chapter describes how to incorporate REXX programming into CM Mouse. The syntax and structure of the REXX programming language is beyond the scope of this book, and it is recommended that you obtain *Procedures Language 2/REXX Reference* which describes the REXX language in OS/2. For the purposes of this document, it is assumed that you are familiar with the REXX language.

CM Mouse allows REXX programs to be executed in response to the user's pressing a mouse button (or combination of buttons) on a host emulation window. This is achieved through the &{rexx...} substitution word (see page "&{rexx PgmSource} or {&rexx PgmSource}" on page 79). REXX programs executed in this way can also execute specific CM Mouse functions through a series of REXX external functions. This two-way communication system between CM Mouse and REXX is collectively referred to as the CM Mouse/REXX interface.

This interface provides great flexibility in how CM Mouse responds to the user's mouse actions. CM Mouse by itself provides a great deal of flexibility by host screen recognition (the SCREEN statement), subdividing screens into "hot spots" (the AREA statement), and an extensive pop-up menu and substitution capability. However, it is sometimes necessary to perform more sophisticated functions which require the power of a full programming language. For example, although CM Mouse can substitute strings from the host screen (using the substitution keywords), it does not have the capability to manipulate that string. Using just CM Mouse keywords, it is not possible to extract substrings, fold the string to upper or lower case, or to separate the string into words and phrases. The REXX interface provides all these functions and many more.

Typical Uses for the REXX Interface

The great majority of button definitions can achieve the desired function without the use of the CM Mouse/REXX interface. However, there are some specific situations in which the interface is very useful:

String Manipulation: REXX has an extensive set of functions for manipulating character strings. It is difficult to do extensive string manipulations with the standard set of CM Mouse substitution words. Thus, if the button definition needs to extract words and phrases from the host screen, to fold strings to upper or lower case, or perform other advanced string functions, then the CM Mouse/REXX interface can provide the necessary function.

Conditional Branching: The CM Mouse button definition language has only limited branching capabilities. A definition can be aborted under a specified condition (if the user cancels a pop-up menu for example), but there are no general branching keywords. The CM Mouse/REXX interface provides access to the full power of REXX structured programming statements such as IF, THEN, ELSE, DO WHILE, and so on.

System Interfaces: CM Mouse provides interfaces to some operating system functions such as task starting with the {sys...} keyword. However it does not provide file services or interfaces to other applications. The CM Mouse/REXX interface provides access to any operating system facilities supported by REXX including file reading and writing, system commands, and alternate command environments using the REXX ‘address’ instruction.

When a particular function can be done with either a REXX program or CM Mouse keywords, it is better to do that function with the keywords. CM Mouse keywords will execute faster and take less storage than the equivalent function written in REXX. Before writing a REXX program to perform a function, carefully examine the set of CM Mouse keywords to see if there is a way they can perform the desired function.

Inline and External REXX Programs

There are two basic forms that a REXX program invoked by CM Mouse can take. The most common form is the “inline” form in which the REXX program statements appear inside the &{rexx...} keyword. In this form, the source text of the REXX program appears in the BDF along with all the other CM Mouse BDF statements and keywords. The following is an example of an inline REXX program as part of a BUTTON statement:

```
BUTTON LEFT "&{rexx +
              if '&{chars 5 1 3}' = 'XYZ' +
                then exit '[pf4]'; +
                else exit '[pf9]'; +
             }"
```

In the second form, “external”, the REXX program resides in a separate OS/2 file. In this case the REXX source text is separate (external) from the BDF. The REXX program is executed with the &{rexx...} keyword as follows:

```
BUTTON LEFT "&{rexx exit PGMNAME(parm1,parm2,...)}"
```

where PGMNAME.CMD is the name of the file which contains the REXX source code. Note that the external REXX program is executed by a single-statement inline program which just returns the EXIT value of the external program as its own. The external program will be located by a search of the directories listed in the OS/2 *PATH* environment variable. Note that this is the PATH value in effect at the time CM Mouse was started.

In general the first (inline) form is preferred for several reasons:

- An inline REXX program executes faster because it is stored in memory when CM Mouse is started and does not have to be fetched from disk to be executed.

- It is easier to read a BDF when the REXX program is inline because the source statements are right there with the rest of the button definition. This makes it easy to determine what a complete button definition does and to change it if necessary. If the REXX program resides in an external file then you must look at two files to understand the whole function of the button definition.
- An inline REXX program is always available, and if the BDF in which it appears is moved, then the REXX program (by definition) moves with it. When a REXX program resides in an external file then you must remember to move it with the BDF if (for example) you send the BDF to someone else to use. It is more difficult to maintain a system of BDFs if the REXX programs reside in separate files.
- Inline REXX programs can have CM Mouse substitutions imbedded within them. This can make it very easy to build REXX programs which use CM Mouse substituted values. External REXX programs cannot have imbedded CM Mouse substitutions. (See “CM Mouse Substitutions in REXX Programs” on page 87.) The first example shown above uses the `&{chars...}` substitution keyword within the source text of the REXX program.

. There are some cases when an external REXX program may be preferred:

- If the REXX program is very long or complex it may take up an inordinate amount of space in the BDF. The BDF may be easier to read and maintain if the REXX program is kept in a separate file.
- If the REXX program is changed often, it may be easier to keep in an external file because CM Mouse will not have to re-read all the BDFs to pick up changes in the REXX program.
- If the same REXX program is used in a lot of button definitions, it may be more efficient and easier to maintain a single external file with the REXX program than to duplicate the program wherever it is needed.

Syntax

REXX programs can only be invoked by CM Mouse through the use of the &{rexx...} keyword. Because this is a CM Mouse substitution keyword, it follows all the normal CM Mouse substitution rules (see “CM Mouse Substitution Words” on page 70 and “Presubstitutions and Runtime Substitutions” on page 80). The substituted value of the &{rexx...} keyword is the EXIT value of the REXX program. If the REXX program does not specify an EXIT value, then a null string is substituted. Consider the following examples:

```
&{rexx exit '[pf4]'}
```

This &{rexx...} keyword will always substitute the string “[pf4]”. It is not a very useful REXX program since it would be easier to just code the string “[pf4]” explicitly. A more realistic example might be:

```
&{rexx if &{hrow}=2 then exit '[tab]'}
```

This example will substitute the string “[tab]” if the host cursor is on row number 2 of the host screen; otherwise, it will substitute a null string.

External REXX programs must follow the normal REXX syntax rules. External REXX programs cannot contain CM Mouse substitution keywords. Note, however, that substituted values can be passed to external REXX programs when they are invoked. For example:

```
&{rexx exit mypgm('&{chars 2 1 3}', &{mrow}, &{mcol})}
```

This would run the external REXX program MYPGM.CMD and would pass it three parameters (the first 3 characters of the 2nd line of the host screen, and the row and column position of the mouse cursor).

Inline REXX programs must also follow REXX syntax rules, and in addition, must conform to CM Mouse button definition syntax rules. As part of a button definition, inline REXX programs can contain CM Mouse pre- and runtime substitutions as described in "Presubstitutions and Runtime Substitutions" on page 80. The source code in an &{rexx...} keyword is treated the same as any other CM Mouse substitution keyword. As such, it must follow CM Mouse keyword syntax, in particular:

- It must end in a closing brace to delimit the &{rexx...} keyword.
- It may contain any pre- or run-time CM Mouse substitutions. Note, however, that this is a substitution keyword itself so nested substitution rules apply (for example, a presubstitution &{rexx...} keyword cannot contain a runtime substitution).
- It may be continued over multiple lines only by use of the CM Mouse continuation characters "+" and "-".
- If the program exceed 512 characters in length, the &{break} keyword must be used to break up the program into sections of 512 characters or less. If a program exceeds 512 characters, the REXX error message:

  ```
  REXX003: Program is unreadable
  ```

  will be displayed. The &{break} keyword can appear anywhere a normal REXX line break is allowed.

Because the entire CM Mouse keyword is considered a single entity, line breaks are not seen by the REXX interpreter. Thus there are no implied semicolons in an inline REXX program -- you must explicitly code semicolons at the end of REXX statements, even if they are at the end of a button definition line. For example, the following is incorrect:

```
&{rexx +
  a = 'abc' +
  b = 'def' +
  exit a}
```

This will fail to run because the two assignment statements are not separated. After the lines are concatenated, the button definition would read:

```
&{rexx a = 'abc' b = 'def' exit a}
```

which is incorrect REXX syntax. This inline REXX program should be written:

```
&{rexx +
  a = 'abc'; +
  b = 'def'; +
  exit a}
```

A semicolon is not required after the last REXX statement. Note that each line of the source code must end in a CM Mouse continuation character.

Because CM Mouse does not perform any semantic analysis of the REXX program, it can misinterpret string constants within the REXX program which appear to be CM Mouse delimiters. For example:

```
&{rexx +
  a = '}'; +
  exit '}
```

CM Mouse will interpret the brace inside the REXX program as the closing brace of the &{rexx...} keyword. REXX programs which contain CM Mouse delimiters must be placed in a CMD file and run as an external REXX program or must use the REXX “X2C” or “D2C” functions to encode the delimiter.

CM Mouse Substitutions in REXX Programs

As discussed in the previous section, inline REXX programs can contain CM Mouse substitution keywords as part of the REXX source code. Since these substitutions appear within the &{rexx...} substitution keyword, they are treated like any other CM Mouse nested substitution. That is, if the keyword is a *presubstitution* then it is replaced with its substituted value before any part of the button definition is executed. If the keyword is a *runtime* substitution then its value is substituted when the &{rexx...} keyword is executed. Runtime substitutions within REXX programs are all substituted prior to executing any part of the REXX program.

Because CM Mouse does not analyze the semantics of inline REXX programs, string constants which contain CM Mouse substitution characters may be misinterpreted as substitutions. For example, the following REXX program attempts to build a string which contains the characters `&{mrow}`:

```
&{rexx +
  ...                          +
  mystr = '&{mrow}[enter]';  +
  ...                          +
}
```

The variable MYSTR would contain a string which consists of the row number of the mouse cursor followed by the string `[enter]`. Since `&{mrow}` appears to be a CM Mouse substitution, CM Mouse makes the substitution before the REXX program is executed. Now consider:

```
&{rexx +
  ...                              +
  mystr = '&'||'{mrow}[enter]';  +
  ...                              +
}
```

In this example, the REXX variable MYSTR would contain the string `&{mrow}[enter]`. Note that the apparent substitution is not evaluated because the “&” and “{” were separated in the source code and thus CM Mouse does not attempt to perform a substitution. This can be most important when building button definition strings for execution via the *CmmExec* function (see “CmmExec” on page 91).

Debugging REXX Programs

To aid in the debugging of complex REXX programs (inline or external), CM Mouse allows REXX programs to display debug messages in a Presentation Manager dialog box. To display a message, a REXX program can use the REXX ‘SAY’ instruction. CM Mouse will take the string specified and display it in a simple dialog box with an **OK** button. When the user selects the button (or presses **ENTER**) execution of the REXX program continues with the next statement. For example:

```
&{rexx +
       myvar = '&{chars 6 1 10}'; +
       say myvar;                             +
       myvar = myvar||' COMMAND';             +
       say myvar;                             +
       exit ';                               +
     }
```

This program will first display a message box which contains the first 10 characters of the 6th row of the host screen. When the user clicks on the **OK** button of the dialog, another box is displayed with the same data followed by the word `COMMAND`. When that dialog box is dismissed, the program exits.

Syntax errors in REXX programs will not be discovered until the program is executed. Inline programs will be halted and the offending statement displayed along with a REXX error message. External REXX programs will be halted, and the REXX error message:

```
REXX040: Incorrect call to routine
```

will be displayed. To isolate the exact cause of a syntax error in an external REXX program, the REXX ‘SIGNAL’ facility can be used. Insert the statement ‘SIGNAL ON SYNTAX’ at the start of the program, and the following at the end of the program:

```
syntax:
  errormsg='REXX syntax error in line ' sigl':' errortext(rc);
  say errormsg;
  say sourceline(sigl);
  exit ';
```

For example, an external REXX program might be written as follows:

```
/* REXX program */

signal on syntax;     /* Trap syntax errors */

if xyz else do;
exit 'abc';

syntax:               /* Execute this if syntax error */

  errormsg='REXX syntax error in line ' sigl':' errortext(rc);
  say errormsg;
  say sourceline(sigl);
  exit ';
```

When the IF statement is analyzed by the REXX interpreter, a syntax error condition is raised. This causes the code after the SYNTAX label to be executed. The error routine will generate two error messages indicating the type and location of the error.

Note that there is no need to trap syntax errors in inline REXX programs. CM Mouse will automatically trap syntax errors in inline programs and display the REXX error message and the offending source statement.

REXX External Functions

When a REXX program is executed via the CM Mouse &{rexx...} keyword, CM Mouse makes a set of external functions available to the executing REXX program. These functions allow the REXX program to perform a wide range of tasks related to the host session, including popping up menus, presenting input dialogs, and executing button definition strings.

These functions will be automatically known to the REXX program; there is no need for REXX ‘ADDRESS’ instructions to use these functions. Each of the external REXX functions is described in the following sections. Note carefully the syntax of each function; some functions return a value (and are used as expressions in the REXX program), and some return no value and should be invoked with the REXX ‘CALL’ instruction.

CmmSearch

```
Syntax:  Call CmmSearch 'FOR /title/ AT r1 c1 r2 c2 WAIT n NOT';
```

This function allows a REXX program to easily search for a string in the host screen without having to get the screen image through CmmScreen and searching it line-by-line. The parameters for CmmSearch are identical to the {search} keyword described on page “{search FOR ’string’ AT r1 c1 r2 c2 WAIT n NOT ASIS NOQUIT}” on page 64.

If the search is successful, the following occurs:

- The REXX variable *CmmRC* is set to zero.
- The CM Mouse system variables SYSTEM_SROW and SYSTEM_SCOL are set to the row and column in which the string was found.

If the search fails, the REXX variable *CmmRC* is set to a non-zero value.

For example, to search for the string `Hello` anywhere on line 6 of the host screen, the following REXX program could be used:

```
call CmmSearch 'for .Hello. at 6 1 6 0 wait 5';
if CmmRC = 0
  then say 'Hello was found in column ' CmmGet('system_scol');
  else say 'After 5 seconds, Hello was not found';
```

CmmGetScreen

```
Syntax:  Call CmmGetScreen;

               -or-

         Call CmmGetScreen 'n';
```

This function will read the host screen and return it as a series of strings in the REXX compound symbol SCREEN. Each line of the host screen will read into one component of the stemmed variable as follows:

```
SCREEN.1  =  Line #1 of the host screen
SCREEN.2  =  Line #2 of the host screen
...
SCREEN.n  =  Line #n of the host screen
```

In addition, the SCREEN.0 component is assigned a string consisting of the following three values separated by blanks:

```
Session ID (1 char)
Number of rows in host screen
Number of columns in host screen
```

Optionally, a line number can be supplied as a parameter in which case only the specified line will be read. For example, the following would read the entire screen and then examine the line the mouse cursor is on to determine if it starts with the word `Ready;`:

```
&{rexx +
   Call CmmGetScreen;+
   if word(screen.&{mrow},1) = 'Ready;'+
     then ...+
 }
```

The next example reads the 2nd line of the screen, and checks to see if it contains only the characters `XYZ`. Note that the screen data must be stripped of blanks; this function always assigns strings which are the full width of the host screen, including leading and trailing blanks. This example also extracts the session ID, and the number of rows and columns in the host screen.

```
&{rexx +
   Call CmmGetScreen '2';+
   if strip(screen.2, 'B') = 'XYZ'+
     then ...+
   parse var screen.0 session rows cols;+
 }
```

Note

If a line number parameter is supplied and it is less than 1 or greater than the number of host screen lines, a runtime error is flagged. The value supplied cannot be zero or negative to indicate an offset from the end of the screen.

CmmInfo

```
Syntax:  var = CmmInfo();
```

This function returns a string containing information about the status of the current host session. The returned string contains 8 values separated by blanks. Note that the “()” is required after the function name. The returned string contains:

```
Session ID (1 character)
Host Cursor Row
Host Cursor Column
Mouse Cursor Row
Mouse Cursor Column
Number of Host Rows
Number of Host Columns
Type of host session (3270/5250)
User Text Editor
```

For example:

```
 &{rexx +
    Data = CmmInfo();+
    parse var Data id hrow hcol mrow mcol maxrow maxcol type editor;+
    if maxrow > 24 then ...;+
  }
```

Note that most of the information returned is available to inline REXX programs through substitutions (for example, `&{hrow}`, `&{hcol}`, and so on).

CmmExec

```
Syntax:  Call CmmExec 'ButtonDefinitionString';
```

This function allows a REXX program to execute a button definition string. The string can contain any CM Mouse keywords, including pre- and runtime substitutions and host keystrokes. The string is executed by CM Mouse just as if it had come from a button definition file.

The REXX variable *CmmRC* will be set to indicate the success or failure of the execution. A button definition which is aborted by a cancelled pop-up menu, a cancelled input dialog, a failed {search...} keyword, or by an invalid CM Mouse keyword is considered to have “failed”.

The following example would send a PF6 key to the host, and then pop up a confirmation dialog with the question “Really do this thing?” The program then checks to see how the user responded (**OK** or **CANCEL** button).

```
&{rexx +
   Call CmmExec '[pf6]{?Really do this thing?}';+
   if CmmRC<>0                             +
     then .... /* User cancelled */        +
     else .... /* User confirmed */        +
  }
```

Note that CM Mouse presubstitutions which appear in the *CmmExec* parameter will be substituted before the REXX program starts. This can lead to some unexpected results:

```
&{rexx +
   Call CmmExec '{rowcol 5 10}';+
   Call CmmExec 'Host cursor is now in row &{hrow}';+
}
```

You might expect this program to move the host cursor to row 5 column 10, then type on the host screen `Host cursor is now in row 5`. However, the &{hrow}

substitution is done *before* the REXX program is executed, so the value is whatever row the cursor was in before the REXX program started.

You can do a presubstitution in a CmmExec call by using REXX strings to prevent the keyword from appearing like a presubstitution:

```
&{rexx +
   Call CmmExec '{rowcol 5 10}';+
   Call CmmExec 'Host cursor is now in row &'||'{hrow}';+
}
```

This example will display `Host cursor is now in row 5` because CM Mouse does not see any presubstitution within the &{rexx...} keyword. REXX string concatenation has been used to hide the substitution until the REXX program is actually executed. The same technique can be used to hide runtime substitutions if necessary. See “CM Mouse Substitutions in REXX Programs” on page 87.

CmmConnect

```
Syntax:  Call CmmConnect 'c';
```

This function allows a REXX program to disconnect from the current host session, and optionally connect to another host session. A single character argument must be supplied. The argument must be the session ID of a currently valid host session, or the special value ’*’. If a valid session ID is supplied, CM Mouse will disconnect from the current session, and connect to the specified session. All host interactions thereafter will be with the new host session. If the value ’*’ is specified, the current host session is disconnected and no other session will be connected. The session ID parameter is not case sensitive.

The REXX variable *CmmRC* will be set to indicate the success or failure of the connect attempt. A non-zero value of *CmmRC* indicates that the specified host session could not be connected (and the previous session, if one, is still connected).

This function does not cause the host session window to be activated or maximized.

This function can be used in a REXX program which (by some means other than CM Mouse) uses the EHLLAPI host interface. Because the EHLLAPI host interface allows only one program to have the session open at a time, CM Mouse must be disconnected from a host session before any other program can use the interface. For example:

```
(Continuation characters "+" omitted for clarity)

&{rexx
    Call CmmConnect '*';       /* Disconnect */
    ...
    ...                        /* EHLLAPI interactions */
    ...
    Call CmmConnect 'b';       /* Connect to B */
    Call CmmInfo();            /* Get information on B session */
    ...
    ...
    Call CmmConnect '&{sid}';  /* Reconnect to original session */
    if CmmRC<>0
      then...                  /* Connect failed... */
 }
```

Note that the final *CmmConnect* uses the &{sid} substitution which is done *before* the REXX program is executed, and thus substitutes the session which was active at the time the REXX program started.

CmmPopup

```
Syntax:  var = CmmPopup('PopupName');
```

This function will cause CM Mouse to display the named pop-up menu. The returned value is the string returned by the menu (for example, the button definition string on the menu LINE chosen by the user). If the user cancels the menu, then a null string is returned and the REXX variable *CmmRC* is set to a non-zero value.

Note that the string returned by the menu is not executed or interpreted by CM Mouse in any way. No substitutions are done on it. The string is returned exactly as it appears in the menu file.

Consider the difference between:

```
&{rexx +
    Call CmmExec '&'||'{popup mymenu}';+
 }
```

and

```
&{rexx +
    data = CmmPopup('mymenu');+
 }
```

Both of these REXX programs cause the MYMENU.MMM menu to be displayed. The first program also executes the string resulting from the menu. The second program does not execute anything, it simply assigns the string to a REXX variable. The second program could be modified to perform identically to the first:

```
&{rexx +
    data = CmmPopup('mymenu');+
    Call CmmExec(data);+
 }
```

It is important to understand that the **CmmPopup** function does not cause any keystrokes to be sent to the host; it just returns the string from the menu LINE the user picked on the pop-up. The REXX program can then determine how to process the string.

CmmPrompt

```
Syntax:  var = CmmPrompt('Question|Answer');
```

This function will cause CM Mouse to display a user-input dialog. The format of the parameter string is the same as the &{?....} keyword (see page "&{?<Qtext>|<Atext>} or {&?<Qtext>|<Atext>}" on page 77). It consists of two strings separated by a vertical bar character. The first string is the title of the dialog box, the second is used to preassign the input field. The leading ″$″ or ″%″ flags may be used in the `Answer` string as in the &{?...} keyword.

The result of the function is the character string in the input field when the user selects the **OK** button. If the user cancels the dialog, the result is a null string and the REXX variable *CmmRC* is set to a non-zero value.

In the following example, the user is prompted for a PC file name. If the file does not exist, the user is reprompted for it. When a valid name is entered, the loop terminates with a message.

```
&{rexx +
   Prompt='Enter PC file name|';+
   Fname =';+
   do until (exist <> '');+
     Fname = CmmPrompt(Prompt||Fname);+
     if CmmRC<>0 then exit;+
     exist = stream(fname,'c','query exists');+
     if exist='' then Prompt='File not found - try again|';+
   end;+
   say 'File' fname 'found.';+
  }
```

Note that if the user cancels the input dialog, the REXX program exits without displaying a message.

CmmGet

```
Syntax:  var = CmmGet('VarName');
```

This function will return the value of a CM Mouse variable (see “Chapter 11. CM Mouse Variables” on page 105). The parameter supplied is the name of the variable who’s value is to be returned. If the specified variable does not exist, a null string is returned. For example, the following would display a message box with the string `MyVar = Hello There`:

```
{set xyz Hello There}{&rexx+
                       myvar = CmmGet('xyz');+
                       say 'MyVar = 'xyz;+
                   }
```

Note the use of runtime substitution to delay execution of the REXX program until after the {set...} keyword is executed.

There is no direct association between CM Mouse variables and REXX variables. That is, CM Mouse variables are not known to REXX, and REXX variables are not known to CM Mouse. The *CmmGet* and *CmmPut* functions can be used to read and write CM Mouse variables from a REXX program.

CmmPut

```
Syntax:  Call CmmPut 'VarName', 'VarValue';
```

This function will assign a new value to a CM Mouse variable (see “Chapter 11. CM Mouse Variables” on page 105). This function requires two parameters; the first is the name of the CM Mouse variable (case insensitive), and the second is the string to be assigned to that variable.

If the variable already exists in CM Mouse, its value is replaced. If it does not exist, it is created. For example, the following would display two message boxes. The first would be displayed with the message `MyVar = Hello There`, and the second would show `MyVar = New value`.

```
{set xyz Hello There}{&rexx+
                       myvar = CmmGet('xyz');+
                       say 'MyVar = 'xyz;+
                       +
    Call CmmPut 'xyz', 'New value';+
    myvar = CmmGet('xyz');+
    say 'MyVar = 'xyz;+
}
```

The *CmmGet/CmmPut* functions can be used to save information from one REXX invocation to another. They can also be used to provide information to other button definitions which use the &{var ...} keyword.

There is no direct association between CM Mouse variables and REXX variables. That is, CM Mouse variables are not known to REXX, and REXX variables are not known to CM Mouse. The *CmmGet* and *CmmPut* functions can be used to read and write CM Mouse variables from a REXX program.

Chapter 10. Drag/Drop Features

OS/2 versus Windows Drag/Drop

The CM Mouse drag and drop features are essentially the same for OS/2 and Windows. There are some operational differences however because the OS/2 desktop (Workplace Shell) and the Windows desktop (Program Manager) are quite different in capability and function.

For the OS/2 environment, objects can be dragged directly from the host screen to the WPS desktop, file folders, shredders, or printers. File objects on the desktop or in folders may be dragged directly to the host screen for uploading.

For the Windows environment, the *CM Mouse Drag/Drop* application window provides the desktop drag/drop point. This program must be running to provide CM Mouse drag/drop file transfer or shredding. The Drag/Drop application provides a means to navigate the disks and directories of the file system and display lists of files. This file list can be the source of a drag to the host screen by selecting the file(s) to be uploaded and dragging them with the *right* mouse button to the host window. Host objects may be dropped on the file list to download files to the selected disk and directory. The Drag/Drop application also provides a shredder on which host objects may be dropped. (As a convenience, files may also be dragged from the file list to the shredder to deleted them).

CM Mouse also supports files dragged *from* the Windows File Manager program to the host screen. However, host object may not be dropped on the File Manager. Thus only file uploading is supported via the File Manager, not downloading, deleting, or printing.

How CM Mouse Drag/Drop Works

The CM Mouse drag/drop features allow CM Mouse scripts to enable and control the drag of host objects to the OS/2 Workplace Shell and the Windows CM Mouse Drag/Drop application window. It also enables the drop of file objects on the host. Like other CM Mouse functions, the drag/drop features are based on a *context sensitive* understanding of the host screens. Only drag/drop functions which make sense in the context of the host applications are allowed.

The basis for CM Mouse drag/drop functions are the BDF *DRAG* and *DROP* statements. These statements determine where dragging and dropping are allowed (what host screens, and what areas of what host screens). In the same way that BUTTON statements define the action to occur when a button is clicked in a particular area of a host screen, the DRAG/DROP statements define the actions to occur when a drag or drop occurs in a particular area of a host screen.

DRAG and DROP statements may occur anywhere in a BDF that the BUTTON statement is allowed. The position of the DRAG/DROP statements within SCREENs and AREAs determine what type of drag/drop functions are supported on a specific host application screen. For example, consider the following BDF fragment:

```
Line      Statement
----      ---------------------------------------------------
 1        SCREEN 1 1 exact * "File List"
 2        DROP FILE "{xfer drop ...}"
 3          AREA 1 1 2 80
```

```
4           DRAG FILE "{xfer drag ... method=screen ...}"
5         AREA 3 1 23 80
6           DRAG FILE "{xfer drag ...}"
7           DRAG DISCARD "{seek}erase[enter]"
```

When this particular application screen is displayed, the following drag/drop functions are defined:

- The user may drop a file from the PC to any part of the host screen. This is allowed because of the DROP FILE statement in line 2 which applies to the entire screen. When the user drops a file anywhere on the screen the script defined in line 2 will be executed.
- The user may drag from anywhere in the first two lines of the host screen to any WPS folder, the OS/2 desktop, or the Windows Drag/Drop file list. This is enabled by line 4 of the script. When the user drops on the PC the script in line 4 is executed. In this case a screen capture will be written to the target file.
- The user may drag from anywhere in the 3rd through 23rd lines of the screen to any WPS folder, desktop, or shredder object. This is enabled by lines 6 and 7. If the user drops on a folder or the WPS desktop then the script in line 6 is executed. If the user drops on a shredder then the script in line 7 is executed.
- The user cannot drag from the first two lines to a shredder object because there is no DRAG DISCARD statement associated with that area of the host screen. If the user begins a drag in this area, the do-not-drop symbol will be displayed when the drag icon is moved over the shredder.
- The user cannot begin a drag operation in line 24 of the host screen because there is no DRAG statement for this area.
- The user cannot drag from any part of the host screen to a WPS printer object because there is no DRAG PRINT statement associated with any part of this screen.

DRAG Statements

A DRAG statement indicates the area of the screen in which the user may originate a drag operation. There are three possible forms of the DRAG statement:

```
DRAG FILE "script"
DRAG PRINT "script"
DRAG DISCARD "script"
```

The first parameter of the DRAG statement defines what type of object may be the target of the drag. The FILE type indicates that the drag which originates in this area may drop on a WPS folder, the OS/2 desktop, or the Window Drag/Drop file list. The PRINT type indicates that the user may drop on an WPS printer object. (Note: DRAG PRINT statements are ignored in Windows). The DISCARD type indicates the user may drop on a shredder object.

If an area of the screen has more than one DRAG statement associated with it, then the user may drop on any of the DRAG types for which there is a DRAG statement. For example, if an area has a DRAG PRINT statement and a DRAG DISCARD statement, then the user may drag to a printer or a shredder. However they would not be able to drop on a folder or the WPS desktop (since there is no DRAG FILE statement).

When the drop occurs, the script associated with the DRAG statement is executed. The script may contain any CM Mouse keyword or function and is executed just like BUTTON statement scripts, with the following exception:

Note

A DRAG FILE statement *must* contain the CM Mouse file transfer keyword:

```
{xfer drag ...}
```

Note that DRAG must be the first parameter of the transfer keyword. The transfer keyword HOST and PC parameters *must not* contain any CM Mouse keyword which would cause interaction with the user. For example, the following is invalid:

```
DRAG FILE "{xfer drag host='&{chars 5 6 12}' pc='&{?Enter PC file name}'}
```

Prompting keywords may be used elsewhere in the script.

DROP Statements

A DROP statement indicates that the area of the screen in which the user may drop an object. The form of the DROP statement is:

```
DROP FILE "script"
```

The first parameter of the DROP statement defines what type of object may be dropped on the host screen. Currently CM Mouse supports only the dropping of FILE objects.

When the drop occurs on the host screen, the script associated with the DROP statement is executed. The script may contain any CM Mouse keyword or function and is executed just like BUTTON statement scripts.

If there is no DROP associated with a particular area of the host screen then the do-not-drop symbol will be displayed if a file is dragged over that area.

The XFER Keyword

The file transfer keyword may be used in DRAG/DROP scripts or in any script to initiate a file transfer. Execution of the script stops until the file transfer is completed, and then resumes with the next keyword in the script. The system variable SYSTEM_XRC will contain ″0″ (zero) if the transfer was successful, or non-zero if it failed.

There are four basic forms of the XFER keyword depending on the context in which it is used:

```
{xfer DRAG ...}
{xfer DROP ...}
{xfer SEND ...}
{xfer RECV ...}
```

The DRAG/DROP forms must be used when the XFER keyword is used in a DRAG/DROP BDF statement. The SEND/RECV forms are used when the XFER keyword is used in a CM Mouse menu or BUTTON script.

Each form has different requirements and rules concerning the specification of the host and PC filenames which are to be used in the transfer. Also some parameters are valid only when used in particular forms. The full XFER keyword syntax is as follows:

```
►►──{──XFER──┬─SEND─┬──TYPE=──┬─VM──┬──PC=──┬─PCFileName─┬──HOST=──────────────►
             ├─RECV─┤         └─MVS─┘       └─*──────────┘
             ├─DRAG─┤
             └─DROP─┘

►──┬─HostFileName─┬──┬─────────────────┬──┬─────────────────────────┬──────────►
   └─*────────────┘  └─SESS=──┬─A───┬─┘  └─XLATE=──┬─*─────────┬─┘
                              ├─B───┤              └─XlateName─┘
                              ├─...─┤
                              └─*───┘

►──┬─────────────────────────┬──┬────────────────────┬──┬────────────────────────┬──►
   └─METHOD=──┬─EHLLAPI─┬─┘  └─QUIET=──┬─YES─┬─┘  └─CLEAR=──┬─YES─────┬─┘
              └─SCREEN──┘              └─NO──┘              ├─NO──────┤
                                                            └─DEFAULT─┘

►──┬───────────────────────────┬──}──────────────────────────────────────────►◄
   └─PARMS=──XferParameters──┘
```

The parameters have the following meanings:

SEND Transfers the specified PC file to the host. The PC file name must be fully qualified with drive, path, and name. The host file name may be explicitly given or ″*″ may be used for automatic name mapping (see "Automatic Name Mapping" on page 102). The METHOD option is not allowed (SCREEN based transfer to the host is not supported).

RECV Transfers the specified host file to the PC. The host file name must be given. The PC file name may be fully qualified with a drive, path, and name, or the ″*″ may be used in place of the name for automatic name mapping (see "Automatic Name Mapping" on page 102).

DRAG Transfers the specified host file to the PC as part of a DRAG FILE statement. This form may not be used except as part of a DRAG FILE statement in a BDF. The host file name must be be specified (possibly by substitution of data from the host screen). The PC file name must be either a simple file name (no drive or path), or an ″*″ for automatic name mapping. The drive and path of the file is determined by where the drop occurred in the OS/2 WPS (or the current drive/directory shown in the Windows Drag/Drop program).

DROP Transfers the specified PC file to the host as part of a DROP FILE statement. This form may not be used except as part of a DROP FILE statement in a BDF. The host file name may be explicitly given or ″*″ may be used for automatic name mapping. The PC file name must be ″*″ and no other value is allowed. The actual file name is determined by the source of the drag operation.

TYPE Specify the type of host for the file transfer. The only supported values are ″VM″ and ″MVS″. *This parameter is required.*

PC Specify the name of the PC file for the transfer. The name must be enclosed in a single delimiter character which does not occur in the file name itself. See the SEND/RECV/DRAG/DROP descriptions for the format of this parameter. *This parameter is required.*

HOST Specify the name of the host file for the transfer. The name must be enclosed in a single delimiter character which does not occur in the file name itself. See the SEND/RECV/DRAG/DROP descriptions for the format of this parameter. *This parameter is required.*

SESS Specify the session on which the file transfer is to take place. A value of ″*″ will use the current session. If not specified, ″*″ is used.

METHOD

Specify the method to be used for the data transfer. Currently the only supported values are ″EHLLAPI″ and ″SCREEN″:

EHLLAPI

A file-to-file data transfer is done using the normal emulator file transfer facilities. This is the default.

SCREEN

A screen-to-file data transfer is done. This value is valid only for RECV and DRAG type transfers. The host screen image is captured and written to the PC file. The host filename parameter is used to define what (rectangular) portion of the host screen is to be captured. The format is:

```
HOST='startrow startcol endrow endcol'
```

The ending row/columns can be specified as zero or negative to indicate offsets from the edges of the screen. Currently SEND/DROP type transfers with a SCREEN method are not supported.

XLATE

Specify the symbolic name of a file translation type. A translation type is a shorthand way of defining what file transfer parameters will be used (binary, ascii, etc). Translation types are defined using CM Mouse SET statements in a BDF script. Translation types are SET statements of the form:

```
SET XFER-<method>-<SEND┘RECV>-<type>-<XlateName>
 "<file transfer parameters>"
```

Where:

- <method> is the ″method=″ value in the {xfer} keyword (currently only the ″EHLLAPI″ method is supported).
- <type> is one of ″VM″ for VM hosts, ″MVSSEQ″ for MVS sequential datasets, or ″MVSPDS″ for MVS partitioned datasets.
- <XlateName> is the symbolic name of this translation type.
- <file xfer parms> are the standard command-line parameters for file transfer (such as ″ASCII CRLF″).

For example, the following would define the translation type ″text″ for files sent to VM:

```
SET xfer-ehllapi-send-vm-text "ASCII CRLF"
```

Then any {xfer ... xlate=text} would use the file transfer parameters ″ASCII CRLF″.

If ″*″ is used, the translation type is taken from the automatic name mapping table (e.g. the transfer type will depend on the source file type). See “Automatic Name Mapping”.

QUIET
If ″YES″ is specified, no progress or status window will be displayed during the file transfer. If ″NO″ is specified, a progress window is displayed. Only the first character is required. The default is ″Y″.

CLEAR
If ″YES″ is specified, a 3270 CLEAR key is sent to the host before the file transfer command. If ″NO″ is specified, no CLEAR key is sent. ″DEFAULT″ may be used to default this option (defaults to ″YES″ for VM, ″NO″ for MVS). Only the first character is required. The default value is ″D″.

PARMS
If this parameter is specified, its value will be used for the file transfer parameters, overriding any parameters set by the XLATE value. The value must be enclosed in a single delimiter character which does not occur within the value itself.

Automatic Name Mapping

When the target file name of a file transfer contains an ″*″, CM Mouse uses *automatic name mapping* to generate the target filename. It is also used to determine the translation type when ″xlate=*″ is specified in the file transfer keyword (XFER).

Automatic name mapping makes it possible to map names between PC and host file systems based on the type of file being transferred. Name mapping is based on the source file type where ″type″ is defined as follows:

```
VM:  The file type
MVS: The last qualifier
PC:  The file extension
```

Name mapping tables are defined to CM Mouse through the use of DEFINE statements in a BDF script file. A name mapping table takes the form:

```
DEFINE NAMEMAP <SEND┘RECV>-<VM┘MVS>
  <SourceType>   <TargetType>   <XferType>
  <SourceType>   <TargetType>   <XferType>
  <SourceType>   <TargetType>   <XferType>
  <SourceType>   <TargetType>   <XferType>
ENDDEFINE
```

Thus there are currently a maximum of 4 name mapping tables, two each for VM and MVS. Each table is a list of mappings from a source file type to a target file type, and the symbolic name of the transfer type to be used.

<SourceType> may use a single ″*″ for a wildcard matching character. <TargetType> may use ″=″ to substitute the source type. It may also specify a delete string enclosed in square brackets. If after any substitutions the target contains the delete string, it is deleted from the target. <XferType> is a symbolic transfer type as described in “{xfer ..}” on page 68.

When a target file name contains an ″*″ or ″XLATE=*″ is used in the XFER keyword, the appropriate name mapping table is searched from top to bottom. The first <SourceType> that matches the source file type is used to generate the target file name and translation type. The last <SourceType> in the list *must* be "*" to act as a default if the file does not match any other source type above it.

For example, the following would be a name mapping table for receiving files from the Host to the PC:

```
define namemap send-vm
  bin      bin     binary         (keep "BIN" filetype)
  *bin     =[bin]  binary         (if ends in BIN, use source filetype without BIN)
  script   scr     text           (map SCRIPT to .SCR)
  module   mod     binary         (map MODULE to .MOD)
  list38*  l38     binary         (any LIST38xx file maps to .L38)
  notebook nbk     text
  *        =       binary         (all others use VM filetype truncated to 3 chars)
enddefine
```

The sample script file XFERDEFS.BDF shows a more extensive example of name mapping tables.

Chapter 11. CM Mouse Variables

CM Mouse allows button definitions and BDF/MMM files to set and use the value of "variables." A variable is a string of characters that can be changed by certain CM Mouse keywords. Variables have two parts: a name, which is a way to refer to the variable, and a value, which is a string of characters. Variables do not have to be predefined or "declared" as in traditional programming languages.

A variable name can be any length and can contain any nonblank characters. Variable names are not case sensitive, so the variable name `VARname` refers to the same variable as `varName`. You can type variable names in any way that is most readable to you.

Setting the Value of a Variable

There are two ways to set the value of a variable. You can either use the {set...} keyword in a button definition or assign a value with a SET statement in a BDF or MMM file. The following sections describe these two techniques.

Setting the Value of a Variable in a Button Definition

The value of a variable can be set by the {set...} keyword in a button definition (see "CM Mouse Control Words" on page 56). The keyword format is:

```
{set User-Variable-Name Value}
```

The value starts with the first nonblank character after the name and continues to the closing brace. It can contain any characters and can be formed by explicit characters or substitutions. For example:

```
{set UserName J. Smith}
```

This would set the value of the variable *UserName* to the string *J. Smith*. The value can be substituted by using a CM Mouse substitution keyword:

```
{set ColNum &{hcol}}
```

This would set the variable named *ColNum* to the column number of the host cursor. If the host cursor was in the 10th column of the screen at the time this keyword was executed, then the value of *ColNum* would become the character string *10*. Another example:

```
{set Host-Location The cursor is at &{hrow},&{hcol}}
```

This would set the variable *Host-Location* to the string *"The cursor is at x,y"* where *x* and *y* are the row and column numbers of the host cursor.

```
{set Filename &{?Enter file name}}
```

This example would display an input dialog, prompting the user to enter a file name. Whatever the user typed in the input field would become the value of the variable *Filename*.

```
{set Filename &{?Enter file name|&{var Filename}}}
```

This example is the same as the previous one, except that the default string displayed in the input field will be the current value of the *Filename* variable. This allows you to build input prompts that remember the last value the user entered.

Setting the Value of a Variable in a BDF or MMM File

A variable can also be assigned a value during BDF reading (initialization) and during MMM reading (invoking a pop-up). To understand how this works, it is important to know when and how CM Mouse reads BDF (button definition) and MMM (menu) files. BDF files are read only once, during CM Mouse initialization. MMM files are read each time they are displayed. BDF files can be reread by selecting the *Reread BDF files* from the action bar of the CM Mouse control panel.

The value of a variable can be set in a BDF or MMM file by using the SET statement. The format is:

```
SET Var-Name "Value"
```

The variable value can be formed from explicit characters and from substitutions. (Note that only *presubstitutions* are allowed in SET statements, no runtime substitutions are allowed.) SET statements are only processed when the BDF or MMM file in which they appear is read. Consideration should be given to *when* SET statements are processed when using substitutions. For example, BDF files are read only during initialization, so it would not make much sense to do a substitution involving the cursor position, such as the following:

```
SET CurPos "&{hcol}"
```

In a BDF file, this would save the position of the host cursor at the time CM Mouse was started (or the last time the BDFs were reread). That is generally not very useful. It may be quite useful, however, to have such a statement in an MMM file, since the MMM file is read and processed whenever the pop-up is invoked with the &{popup...} substitution word.

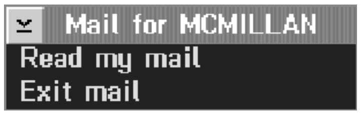
Since a SET statement can contain substitutions, it is possible to cause an input dialog to be presented during BDF or MMM reading. For example, suppose the following SET statement appears in a BDF:

```
SET OV-ID "&{?Enter your OfficeVision User ID}"
```

This would prompt the user for an ID when CM Mouse is started and whenever the BDFs are reread. The variable `OV-ID` could then be used in a number of ways in BDF definitions and MMM files, for example:

```
TITLE "Mail for &{var OV-ID}"
LINE "Read my mail"...
LINE "Exit mail"...
```

This menu file uses the variable in the title to get a pop-up menu like the following:



By placing a SET statement in an MMM file, the user can be prompted with an input dialog immediately before a pop-up menu is shown. The value of that input field can then be used in the menu or the menu's substitution text. Statements in BDF and MMM files are processed sequentially from top to bottom. SET statements should appear in an MMM file before the first usage of the variable being set. For example, the following would *not* work as expected:

```
title "&{var MyTitle}"
set MyTitle "&{?A title, please}"
line...
line...
```

The user would be prompted for a title before the pop-up appears, but the value typed in would not appear in the title area of the pop-up because the substitution was done before the value of the variable was SET. This should be written as follows:

```
set MyTitle &"{?A title, please}"
title "&{var MyTitle}"
line...
line...
```

Variable Substitutions (Using the Value of a Variable)

The value of a variable can be used anywhere any other CM Mouse substitution keyword is valid. Like other substitutions, a variable can be presubstituted, or runtime substituted. The format for variable substitution is:

```
&{var Var-Name}         -- Presubstitution
{&var Var-Name}         -- Runtime substitution
```

The semantics are the same as other substitutions; a presubstitution is done before any part of the button definition is executed and a runtime substitution is done as the definition is executed from left to right. For example, consider a definition such as:

```
{set MyFile &{?Enter file to delete}}delete &{var MyFile}
```

This would not work as expected. To understand why not, consider the order of evaluation. First, all presubstitutions are done (from left to right), so the `Enter file` prompt is presented, and the value given by the user is substituted into the {set...} keyword. Then the &{var...} presubstitution is done, but note that the {set...} keyword has not been executed yet, so the value substituted is whatever the value was previously. Finally, execution starts with the {set...} keyword, and the value supplied by the user is assigned to *MyFile*. The delete command is typed on the host but with the old value of *MyFile*. The correct coding of this is:

```
{set MyFile &{?Enter file to delete}}delete {&var MyFile}
```

Note the ampersand (&) is inside the braces of the variable substitution. This delays the substitution until after the {set...} has been executed and `delete` has been typed on the host screen. It would also be proper for the prompt to be a runtime substitution:

```
{set MyFile {&?Enter file to delete}}delete {&var MyFile}
```

Note the ampersand (&) inside the braces of the file name input prompt. This would behave exactly as the previous example.

Rules of Variables

Variables and their values persist for as long as CM Mouse is executing. There is no way to delete a variable, although it can be assigned a null value. A variable may be used in a substitution before it has been assigned a value. Such a usage will substitute a null string. Thus, in effect, there is no distinction between a null variable value and a variable that has never been assigned a value. This allows a variable to be used to hold the default value of an input dialog without having to explicitly preassign it:

```
{set Target &{?Enter target name|&{var Target}}}
```

This example would prompt for the target name, using the target name that was last entered as the default value. The first time this is executed, the &{var...} substitutes a null because *Target* is not assigned any value yet. The next time, &{var...} will substitute the value the user entered the previous time this definition was executed.

Predefined System Variables

The following variables are automatically set when CM Mouse is started:

Variable Name	Value	Description
SYSTEM_ENV	OS2, WIN, DOS, or UNIX.	Indicates the operating environment.
SYSTEM_SROW	Numeric	Row number of last successful {search} function
SYSTEM_SCOL	Numeric	Column number of last successful {search} function

SYSTEM_EMTYPE	ES, CM, PP, PC, P2, P3	The emulator for which CM Mouse is configured:
		`ES  Extended Services`
		`CM  Communications Manager/2`
		`PC  Personal Communications`
		`PP  Advantis Passport`
		`P2  Personal Communications V2 (Win)`
		`P3  Personal Communications V3+ (Win)`

SYSTEM_XRC	Numeric	Return code from last file transfer ({xfer...} keyword).
SYSTEM_HOSTFILE	Host File Name	Host file name used in last file transfer after any name mapping was applied.
SYSTEM_PCFILE	PC File Name	PC file name used in last file transfer after any name mapping was applied.
SYSTEM_TMPDIR	PC Directory Path	Path of temp directory used for some drag/drop file transfer operations.

Debugging Hints

CM Mouse has no facilities for directly displaying or modifying user variables. You can easily construct such a facility for your own variables, however, with a pop-up menu like:

```
title "My Variables"
line "MyVar1='&{var MyVar1}'"{set MyVar1 &{?New value|&{var MyVar1}}}"
line "MyVar2='&{var MyVar2}'"{set MyVar2 &{?New value|&{var MyVar2}}}"
line "MyVar3='&{var MyVar3}'"{set MyVar3 &{?New value|&{var MyVar3}}}"
line "MyVar4='&{var MyVar4}'"{set MyVar4 &{?New value|&{var MyVar4}}}"
```

Make this pop-up selectable from the system pop-up CMMOUSE.MMM, and you have a simple display/modify system for your variables. When the menu is displayed, you can see the current values of the variables. Selecting one will display a dialog to change the value with the current value as the default.

Chapter 12. CM Mouse Utility Programs

This chapter describes several utility programs for CM Mouse:

- “The CM Mouse Menu Editor”
- “The CM Mouse Button Simulator” on page 113

The CM Mouse Menu Editor

The CM Mouse Menu Editor provides an easy-to-use WYSIWYG (What You See Is What You Get) editor for CM Mouse pop-up menu files. It will display the menu exactly as it will appear when used by CM Mouse. You can add new items to the menu, delete items, and rearrange them. Using the menu editor, you can create and modify CM Mouse pop-up menu files without knowing anything about the textual file format described in “Chapter 7. CM Mouse Menu Files” on page 45.

The menu editor can run as a standalone application from the OS/2 desktop, or you can invoke it from a pop-up menu’s system menu (see “Pop-up System Menus” on page 13). You can also locate any CM Mouse menu file through desktop folders or the OS/2 drives icon and double-click on it. [5] The menu editor appears similar to that shown in Figure 14 on page 112.

The menu editor shows a representation of the menu nearly identical to how it will look when actually used. There are a few things which are not represented exactly as they will appear:

1. The menu is shown with scroll bars on the right and bottom sides. The actual menu will be sized tall enough for the number of lines it has, and wide enough for the longest line of text. The actual menu will not have scroll bars. The scroll bars are used in the menu editor to keep the representation at a fixed size during editing.
2. The menu is shown with the correct basic colors. However, if colors are specified for an *individual line* of the menu (with the **Color** button), that line will not appear in the specified color.

Anywhere in the menu editor that a list has the text *Drag ’n Drop* near the lower right corner, that list can be reordered by using the right mouse button to drag a list item and drop it elsewhere in the list. When you begin dragging a list item the pointer shape changes to a thick left-arrow. When you release the right button the item is moved to the nearest boundary between two items. Moving the mouse above or below the list while dragging will scroll the list up or down.

5. The ability to double-click on a menu file to open the menu editor requires that the *Install desktop objects* option be selected during CM Mouse installation. An OS/2 Workplace Shell association will be established between *.MMM files and the CM Mouse menu editor program.

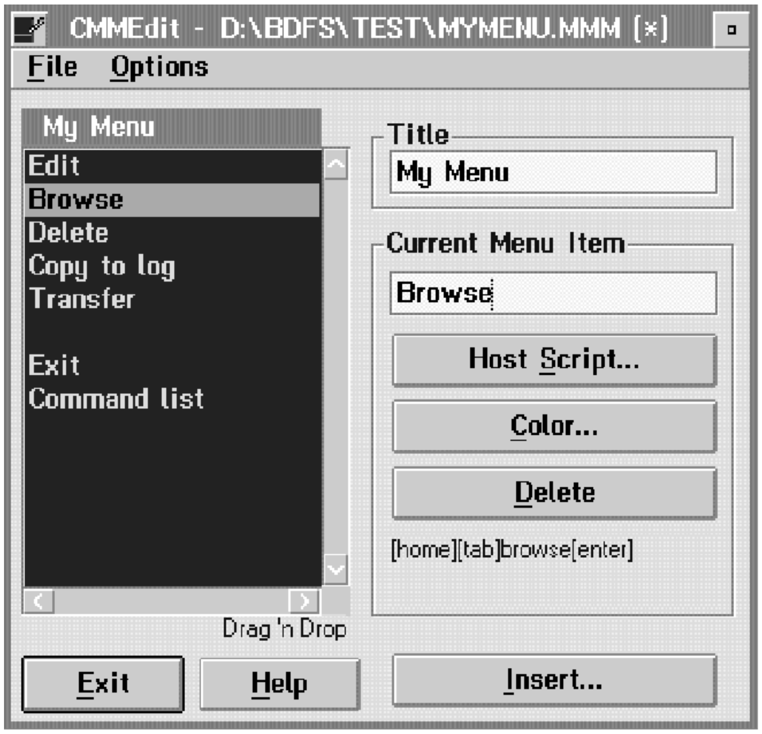


Figure 14. CM Mouse Menu Editor

Menu Title

To change the title of the pop-up menu simply type the desired text in the **Title** field of the main menu editor window. The text shown above the menu will change as you type each character.

Menu Item Text

To change the text displayed for a line of the menu, select the line with the left mouse button. The input field under **Current Menu Item** will be filled with the current text. You can modify the field and the menu will be updated as you type each character.

Menu Item Script

To change the CM Mouse script associated with a menu item, you can either double-click the menu item, or select the item and then press the **Host Script** button. The *Host Script Editor* dialog will be displayed. This dialog allows you to add, delete, and reorder CM Mouse keywords in a script.

Menu Item Color

To change the color of a single line of the menu, select the line and then press the **Color** button. A dialog with 16 color buttons allows you to specify the foreground and background colors for the menu item. Note that the menu displayed in the editor will not show the item colors, but they will appear correctly when the menu is used by CM Mouse.

Delete Menu Item

To delete a line of the menu, select the item and then press the **Delete** button.

Insert Menu Item

To insert a new line of the menu, select the **Insert** button. You can choose to insert the new line before or after the currently selected menu line.

Default Menu Colors

To change the basic menu color scheme, select **Options->Colors** from the menu bar. You can choose new colors for the menu foreground, background, and bounce-bar. When you select **Apply** on the color dialog, the menu display is updated with the new color scheme.

Default Bounce Bar Position

To change the default position of the bounce bar (the initial position it will have when the menu is first displayed), select **Options->Bar Position** from the menu bar. If no bar position is specified, the default of 1 is used.

Menu Placement

To change the position of the menu relative to the host screen, select **Options->Menu Placement** from the menu bar. If no placement is specified, the menu will be placed one row below, an one column to the right of the mouse pointer.

Set Variable Values

To set the value of CM Mouse variables in this menu, select **Options->Set Variables** from the menu bar. A list of all the variables for this menu will be displayed. You can edit the name or value of a variable, add a new variable, delete a variable, or change the order in which they are set.

Exit and Save Menu

To save the changes to the menu and exit the menu exit, press the **Exit** button.

Exit Without Saving

To exit the menu editor without saving changes to the menu, select **File->Quit** from the menu bar. If the menu has been changed, you will be prompted to confirm before the changes are discarded.

The CM Mouse Button Simulator

In some situations it is desirable to cause a CMMouse script to be run without the user clicking on the host screen. For example, you may want to initiate some host interaction from a REXX program, batch file, or desktop object.

Two utility programs provide the ability to “simulate” a CMMouse button click on the current host screen. The simulated click is processed just as if the user had done the click with the mouse.

PM Button Simulator

CMMSIM.EXE is a PM program which will cause a simulated CM Mouse click on a specified host session. If run with no parameters, a dialog is presented on which you can select the host session, the row/column the click should occure in, and the button to be simulated (left, right, left+right, etc). This mode is useful for debugging.

If passed a set of 4 parameters, this program will run the simulated CMMouse button with no user interaction, and then terminate. The parameters are:

```
Session Row Column Button
```

The parameter values are defined as:

Session
: is the session ID on which to simulate a click (A, B, C, etc.).

Row/Column
: are integer values greater than zero which indicate where the simulated click is to occure in the host screen.

Button
: is a CM Mouse button name such at that used in the BDF BUTTON statements (e.g. LEFT, RIGHT, LEFT+RIGHT, DBLLEFT, etc).

Using this utility a desktop program object could be created which would cause some specific CM Mouse functions to be run.

Command Line Button Simulator

CMMSIMC.EXE is a command-line version of the same button simulator described in the previous section. It can be run from the command line or from a REXX program. This program takes the same 4 parameters as the PM button simulator.

In addition, a 5th optional parameter ″WAIT″ can be specified. If WAIT is specified the program will suspend itself until CMMouse is running, has read the script files, and is connected to the host sessions. This can be useful when the simulator is run at the same time CM Mouse is started.

Chapter 13. Button Usage Standards

One of the greatest benefits of using CM Mouse to drive host applications is the consistency it can provide. Many host applications are quite inconsistent in their operation. For example, some host applications use PF3 to cancel a menu and some use PF12. This inconsistency results in keystroke errors and lost productivity. CM Mouse can make inconsistent host applications look and feel similar. With CM Mouse, you do not have to remember whether CANCEL is PF3 or PF12, you just press the RIGHT mouse button.

The advantage that CM Mouse provides is lost, however, if the mouse buttons are customized differently for every host application. For example, suppose CM Mouse were customized so that in OfficeVision the RIGHT button cancelled the current menu, but in FILELIST the RIGHT button deleted a file. This inconsistency could cause you to inadvertently delete files in FILELIST when you wanted to cancel the FILELIST menu.

It is very important then for the BDF writer to customize CM Mouse in a consistent way. Guidelines for customizing CM Mouse have been developed. The standards for each button are described in the following sections. The sample customization files provided with the package adhere to these standards. It may not always be possible to follow the standards exactly, depending on the nature of the host application, but they should be used whenever possible.

LEFT Button Usage

The LEFT mouse button is used to select the item or area under the mouse cursor in a context-sensitive way. To "select" might mean to execute a PF key, execute a command, display a pop-up option list, or display a pop-up command menu. In any case, it is used to initiate an action toward getting some task done on the host. The LEFT button is used to "point-and-shoot" at options or commands on the host screen. It is used to navigate layered menu systems and initiate commands. Its actions are usually dependent on where the mouse cursor is located on the host screen.

RIGHT Button Usage

The RIGHT mouse button is used to cancel the current host menu or back up one menu level. On most OfficeVision/VM screens, the PF12 key performs this function. For other applications, the PF3 key is often used.

On some host application screens, it may not be clear what the user intends with the RIGHT button. For example, on the OfficeVision/VM SEND NOTE screen, PF12 cancels the note. If the user presses the RIGHT button does that mean they intended to cancel the note, or does it mean they have completed the note and want to send it? In cases where the RIGHT button could easily have more than one meaning, the BDF writer may choose to display a pop-up menu to clarify the users intentions. For instance, the RIGHT button may display a pop-up menu with two options on it, *Cancel Note* and *Send Note*.

The function of the RIGHT button is usually not dependent on the position of the mouse cursor.

LEFT+RIGHT Button Usage

For host menus which allow vertical scrolling, this button combination is used to scroll in the downward direction. When the host application allows it, the scrolling should be in pages or half pages. This combination is also used to scroll forward through multi-page menu systems.

If the host menu does not support scrolling, this button combination can be used for some other special purpose.

RIGHT+LEFT Button Usage

For host menus which allow vertical scrolling, this button combination is used to scroll in the upward direction. When the host application allows it, the scrolling should be in pages or half pages. This combination is also used to scroll backward through multi-page menu systems.

If the host menu does not support scrolling, this button combination can be used for some other special purpose.

MIDDLE Button Usage

This button can be used for special purposes, or it can be left to the user to define. The BDF writer should not assume the user has a 3-button mouse.

Chapter 14. Sample Button Definitions and Menus

The following examples give you some idea of what can be done with BDF and MMM files. A number of sample menu files are included in the CM Mouse package, including some similar to these. The CM Mouse package contains sample BDF and MMM files for several popular host applications including PROFS/OfficeVision, RDRLIST, FILELIST, and others. You can change the samples to suit your own taste and preferred style of interacting with the host. You may want to program the mouse for your own host applications, depending on what host applications you use most, your own style of interacting with host applications, and what types of functions you do most often.

The sample button definition files in the CM Mouse package consist of files that are similar to those shown on the next several pages. Definition files for other VM, MVS and AS/400 applications are also included in the sample set. Choose those files that are useful to you, and modify them to fit your own environment and needs. As you create or modify button definition files, consider using the button usage standards as outlined in Chapter 13. Button Usage Standards. Adhering to these standards will ensure that your button definitions are easily used by anyone familiar with CM Mouse.

In general, the more screens a particular host application has, the more statements are required in the button definition file for it. Usually there is a single SCREEN statement for each application menu, and there may be a number of AREAs within each SCREEN. If there are a large number of application menus, there will be a large number of SCREEN (and possibly AREA) statements. It is suggested that you make a separate button definition file for each host application and simply *include* the files in a single master file. This makes it easier to add new definitions for new host applications.

If an application will use a number of pop-up menu (MMM) files, then it is recommended that you use a subdirectory to group the files conveniently in one place. The sample set of definitions has done this -- using separate subdirectories for OfficeVision/VM, for CALLUP, and other applications. This will make it easier to manage the files for a specific application, and will prevent MMM file name conflicts with other applications.

For example, suppose you are creating BDF and MMM files for a host application called MYAPP.

1. Create a new directory under the main CMMOUSE directory:
   ```
   mkdir c:\cmmouse\myapp
   ```
2. Create the button definition file “C:\CMMOUSE\MYAPP\MYAPP.BDF”. This file will contain all the SCREEN, AREA, and BUTTON statements for the application.
3. Add the following statement to the “C:\CMMOUSE\CMMOUSE.BDF” file:
   ```
   INCLUDE MYAPP\MYAPP
   ```
   This tells CM Mouse to load the BDF file “MYAPP.BDF” from the subdirectory MYAPP.
4. Now create all your .MMM (pop-up menu) files in the MYAPP subdirectory. Note that the &{popup...} keyword must explicitly name the subdirectory of the menu file. For example, MYAPP.BDF might have a statement like:
   ```
   BUTTON LEFT "&{popup myapp\mainmenu}"
   ```

By prefixing the pop-up menu name with a relative path name, CM Mouse knows to fetch the menu file from the MYAPP subdirectory.

By putting all the files related to your application into a subdirectory, you can avoid accidentally using the same pop-up menu file name as some other application. Using the example above, if there were two applications which used a pop-up menu named MAINMENU they would both refer to the same physical file. By putting the pop-up menu files for each application in separate subdirectories they can each have their own MAINMENU.MMM files without any conflict.

The SCREEN and AREA statements, along with menus, provide great flexibility in how the mouse will interact with a particular host screen. Use your imagination and knowledge of your own particular work habits to envision how you might like the mouse to interact with a host menu. In particular, look for things that you do often, and see how the mouse and pop-up menus can help you do them faster.

You can use the guidelines described in Chapter 13. Button Usage Standards to get an idea of how the mouse might interact with the host. As a general rule, users of a host application will be most efficient if they can use the mouse for every function they want to perform. Every time they have to return to the keyboard to perform a function, time is lost moving from the mouse to the keyboard and looking for the right keyboard keys. Try to set up CM Mouse so that the mouse can be used for everything and the user does not have to touch the keyboard except for text input (such as typing the text of a document).

Host Application Examples

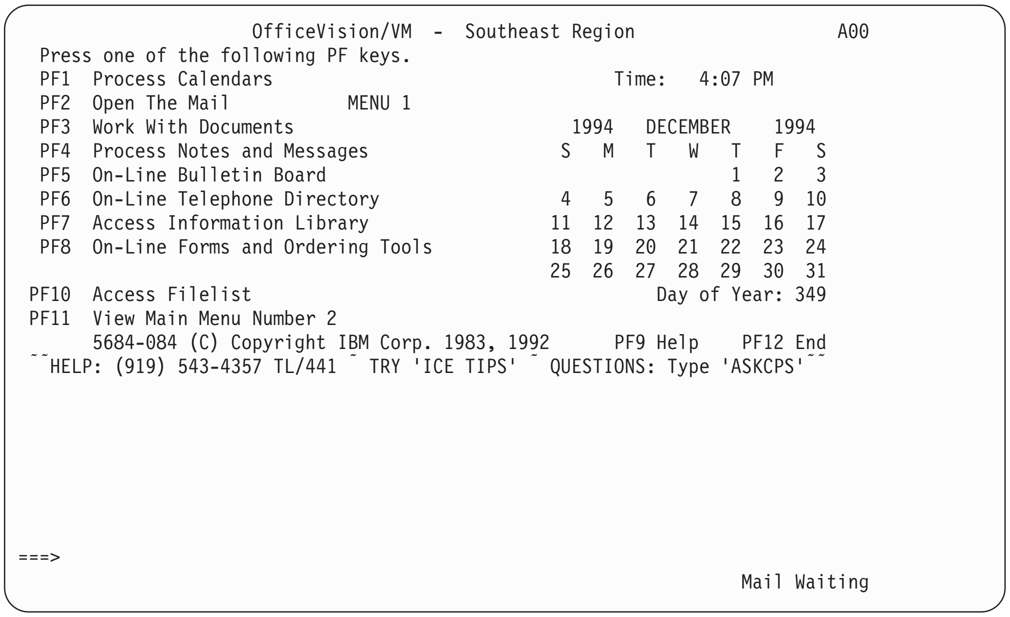
This section illustrates menu file and button definition examples for host applications. The following applications are described:

- PROFS/OfficeVision
- VM RDRLIST
- Text Editors.

The first example is explained in a step-by-step fashion as a button definition file is constructed. These are just a few examples to help you understand the basics of writing a button definition. It is suggested that you examine the sample files supplied with CM Mouse to see how they are constructed and to see more complex examples of button definitions.

PROFS/OfficeVision Examples

If you use PROFS/OfficeVision frequently, you might set up your mouse buttons and pop-up menus to make it faster and easier to use. Assume that the OfficeVision (OV) main menu appears something like that shown below (your local OV installation may be somewhat different):



```
                    OfficeVision/VM  -  Southeast Region                  A00
  Press one of the following PF keys.
  PF1  Process Calendars                               Time:   4:07 PM
  PF2  Open The Mail           MENU 1
  PF3  Work With Documents                         1994   DECEMBER    1994
  PF4  Process Notes and Messages                 S   M   T   W   T   F   S
  PF5  On-Line Bulletin Board                                     1   2   3
  PF6  On-Line Telephone Directory                4   5   6   7   8   9  10
  PF7  Access Information Library                11  12  13  14  15  16  17
  PF8  On-Line Forms and Ordering Tools          18  19  20  21  22  23  24
                                                 25  26  27  28  29  30  31
 PF10  Access Filelist                                   Day of Year: 349
 PF11  View Main Menu Number 2
       5684-084 (C) Copyright IBM Corp. 1983, 1992      PF9 Help    PF12 End
 ~~HELP: (919) 543-4357 TL/441 ~ TRY 'ICE TIPS' ~ QUESTIONS: Type 'ASKCPS'~~






===>
                                                              Mail Waiting
```

Figure 15. Sample OfficeVision Main Menu

Note that OV displays a unique screen identifier in the upper right corner of most menus. We can use this to distinguish the different OV screens. (If the screens did not have that field, we could use the menu titles.) Now, assume you want to use the LEFT button to pick things, the RIGHT button to back up one menu, and the combination buttons to do scrolling (on those menus that allow it). This is consistent with the guidelines described in “Chapter 13. Button Usage Standards” on page 115. Using a consistent set of conventions like this makes it easier to use the mouse in different applications.

To construct a button definition file for the main OV menu, first write a SCREEN statement which will recognize the main menu. The menu can be recognized easily by the screen identifier in the upper right corner (“A00”). The SCREEN statement to recognize this menu would be:

```
SCREEN 1 76 exact * "  A00"  -- OfficeVision main menu
```

This statement tells CM Mouse to look at row 1 column 76 for a string which consists of the characters “A00” with two leading blanks. The leading blanks are included in the search because CM Mouse might find another host menu that has (for example) “XYA00” in the upper right corner. By including the leading blanks you can be sure that this menu is in fact, the OV main menu.

The next thing to add to the button definition file is the default button definitions for this screen. We will defer writing this part for a moment, and just put in a comment as a place holder. We will come back to this part later:

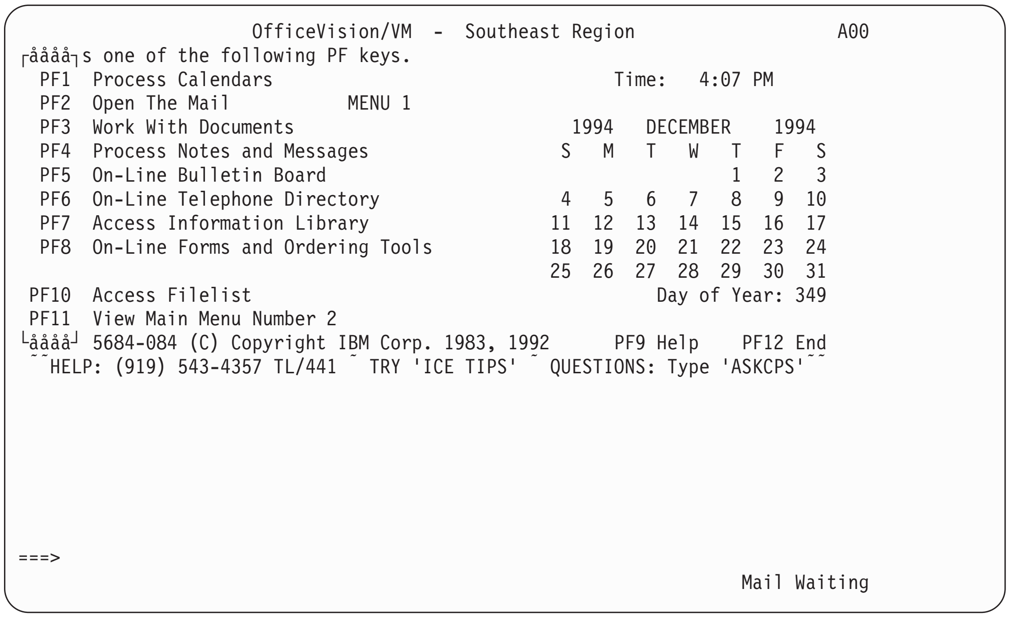
```
SCREEN 1 76 exact * "  A00"  -- OfficeVision main menu
*
* Default buttons will go here...
*
```

Now define the “hot spots” for this screen. In other words, what areas of the screen should have special meaning if the user points and clicks on them? There are some obvious things that might come to mind. For example, the user might want to point-and-shoot at the list of PF keys along the left side of the screen. To allow the user to do this, we define a hot spot (or “AREA” in CM Mouse terminology) which covers the PF keys. Looking at the host screen, the PF key list starts on line 3 and goes though line 13. The “PF...” text starts in column 3 and ends in column 5. (You can easily get the row and column numbers by using the Screen Map “Map Trace” window; see “Map Trace Window” on page 24.)

We could define the hot spot to cover just exactly the area of row 3 column 2 to row 13 column 5. It is a good idea, however, to allow the user some margin when pointing at something. For example, you could allow the user to miss by a character or two on either side of the PF key text. So, define a hot spot which goes from row 3 column 1 to row 13 column 6:

```
SCREEN 1 76 exact * "  A00"  -- OfficeVision main menu
*
* Default buttons will go here...
*
AREA 3 1 13 6  ---- PF keys
```

Figure 16 shows the hot spot we have just defined:



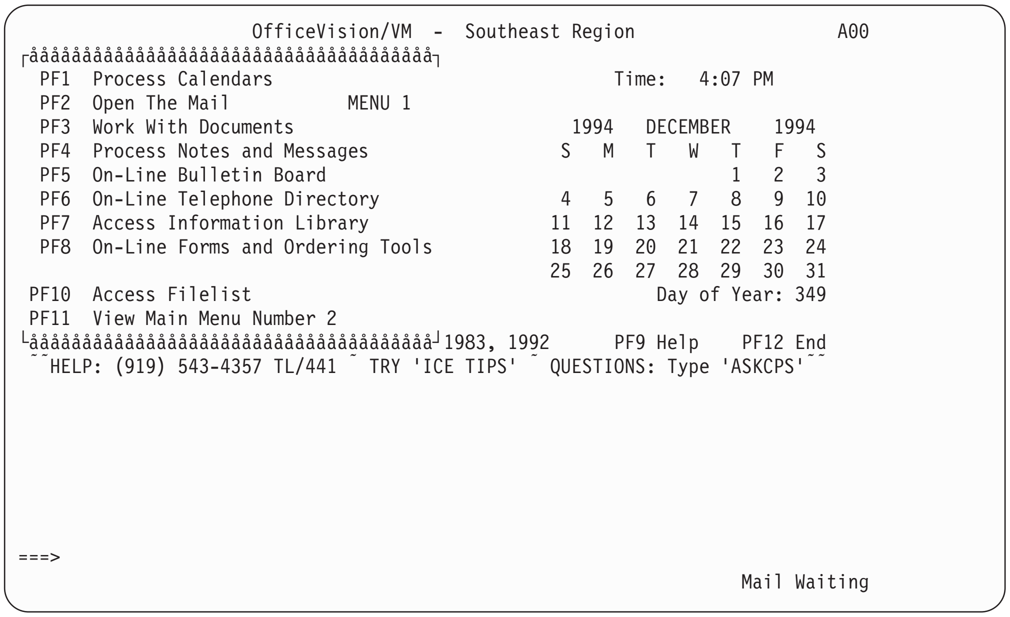
```
                    OfficeVision/VM  -  Southeast Region                  A00
┌åååå┐s one of the following PF keys.
  PF1  Process Calendars                                Time:   4:07 PM
  PF2  Open The Mail           MENU 1
  PF3  Work With Documents                          1994   DECEMBER    1994
  PF4  Process Notes and Messages                  S   M   T   W   T   F   S
  PF5  On-Line Bulletin Board                                      1   2   3
  PF6  On-Line Telephone Directory                 4   5   6   7   8   9  10
  PF7  Access Information Library                 11  12  13  14  15  16  17
  PF8  On-Line Forms and Ordering Tools           18  19  20  21  22  23  24
                                                  25  26  27  28  29  30  31
 PF10  Access Filelist                                        Day of Year: 349
 PF11  View Main Menu Number 2
└åååå┘ 5684-084 (C) Copyright IBM Corp. 1983, 1992      PF9 Help    PF12 End
 ˜˜HELP: (919) 543-4357 TL/441 ˜ TRY 'ICE TIPS' ˜ QUESTIONS: Type 'ASKCPS'˜˜







===>
                                                                  Mail Waiting
```

Figure 16. Sample Hot Spot Definition

Next, tell CM Mouse what buttons are defined in the hot spot. According to our button conventions, the LEFT button is used to point-and-shoot. So we want to define the LEFT button in the hot spot in such a way that when users click on a PF key, the PF key they click on is sent to the host as a keystroke. The {pfkey} keyword (see page “{pfkey}” on page 62) performs this function. We write a BUTTON statement to define the action of the LEFT button in the hot spot:

```
SCREEN 1 76 exact * "  A00"  -- OfficeVision main menu
*
* Default buttons will go here...
*
AREA 3 1 13 6
BUTTON LEFT "{pfkey}"   Execute PF key
```

The text after the second double-quote character is a comment; it is not part of the actual button definition. Now let’s improve on that concept a little bit. We could allow users to point-and-shoot not only at the “PFx” text itself, but at the descriptive text which follows. For example, they could point-and-shoot at “Open the mail”. To allow this, we have to make our hot spot larger. We will expand it to run from row 3 column 1 to row 13 column 40. This covers all the PF keys and their descriptions:



```
                     OfficeVision/VM  -  Southeast Region                  A00
┌åååååååååååååååååååååååååååååååååååååååå┐
  PF1  Process Calendars                                Time:   4:07 PM
  PF2  Open The Mail           MENU 1
  PF3  Work With Documents                          1994   DECEMBER    1994
  PF4  Process Notes and Messages                 S   M   T   W   T   F   S
  PF5  On-Line Bulletin Board                                     1   2   3
  PF6  On-Line Telephone Directory                4   5   6   7   8   9  10
  PF7  Access Information Library                11  12  13  14  15  16  17
  PF8  On-Line Forms and Ordering Tools          18  19  20  21  22  23  24
                                                 25  26  27  28  29  30  31
 PF10  Access Filelist                                      Day of Year: 349
 PF11  View Main Menu Number 2
└åååååååååååååååååååååååååååååååååååååååå┘1983, 1992      PF9 Help    PF12 End
 ¯¯HELP: (919) 543-4357 TL/441 ¯ TRY 'ICE TIPS' ¯ QUESTIONS: Type 'ASKCPS'¯¯







===>
                                                                   Mail Waiting
```

The button definition needs to change slightly to accommodate this. The {pfkey} keyword looks for a PF key number under or near the mouse cursor. However, we are now allowing the user to point a number of characters to the right of the PF key number. By adding the “first” parameter to the {pfkey} keyword, we can tell CM Mouse to look not under the mouse cursor, but to the beginning of the line for the PF key number. Our button definition now reads:

```
SCREEN 1 76 exact * "  A00"  -- OfficeVision main menu
*
* Default buttons will go here...
*
AREA 3 1 13 40
BUTTON LEFT "{pfkey first}"   Execute PF keys
```

You may ask about the blank line between PF8 and PF10 on the menu. Our hot spot includes that line, but it contains no PF key description. We can include that line in our hot spot, because if CM Mouse does not find a PF key number at the beginning of the line the user clicked on, a short beep will be sounded and CM Mouse will not send any keystroke to the host (see the description of the {pfkey} keyword on page “{pfkey}” on page 62).

There are two other PF keys on the main menu the user may want to point at. PF9 and PF12 appear under the calendar on the right side of the screen. We will define another hot spot (AREA) for those PF keys. They are on line 14 from column 57 to 76. Again, we will define some margin for error in the area so the user may click a bit to the left or right of the PF keys:

```
SCREEN 1 76 exact * "  A00"  -- OfficeVision main menu
*
* Default buttons will go here...
*
```

```
AREA 3 1 13 40
BUTTON LEFT "{pfkey first}"   Execute PF keys
AREA 14 55 14 80
BUTTON LEFT "{pfkey}"  Execute PF keys
```

With this button definition, the user can point at any PF key, or any PF key description on the OV main menu and click the LEFT button to execute that PF key function.

To let the user point at other things on the OV main menu, add some more hot spots. Notice that in the lower right corner is a “Mail Waiting” indicator. It would be nice if the user could click on that indicator to read the mail. Also notice that PF2 is the function key we need to send to the host to open the mail. We could define a hot spot for that Mail Waiting indicator as follows:

```
SCREEN 1 76 exact * "  A00"  -- OfficeVision main menu
*
* Default buttons will go here...
*
AREA 3 1 13 40
  BUTTON LEFT "{pfkey first}"   Execute PF keys
AREA 14 55 14 80
  BUTTON LEFT "{pfkey}"   Execute PF keys
AREA 24 67 24 80
  BUTTON LEFT "[pf2]"  Open the mail
```

If the user clicks with the LEFT button on row 24 between columns 67 and 80, then CM Mouse will send a PF2 keystroke to the host. But what happens if the user is running with a 32-line display screen instead of 24? The Mail Waiting indicator will appear on line 32 (the last line of the display); not on line 24. As written above, this button definition file would fail to work as expected on a 32-line host screen (or any host screen that was not 24 lines long).

Since you cannot know ahead of time what size screen the user will be running on, you must write AREA statements that are independent of screen size. You can define an AREA either in terms of absolute row and column numbers as we have been doing, or in terms of an offset from the last line of the screen. For example, in the case of the OV Mail Waiting indicator, we want a hot spot which is on the last line of the screen, no matter what line number that happens to be (24, 32, or something else). You can write such an AREA statement by replacing the ROW numbers with an offset number. An offset number is either zero or a negative number. Zero indicates the last row of the screen. The value minus 1 indicates one line above the last line of the screen (the second line from the bottom). A value of minus 2 is the third line from the last, and so on. Using this offset technique, we can rewrite our button definition file as:

```
SCREEN 1 76 exact * "  A00"  -- OfficeVision main menu
*
* Default buttons will go here...
*
AREA 3 1 13 40
  BUTTON LEFT "{pfkey first}"   Execute PF keys
AREA 14 55 14 80
  BUTTON LEFT "{pfkey}"   Execute PF keys
AREA 0 67 0 80
  BUTTON LEFT "[pf2]"  Open the mail
```

Notice that the last AREA statement now indicates a hot spot which is between columns 67 and 80 of the last line of the screen. This hot spot will always be on the last line of the screen, no matter what size host screen is being used.

Similarly, we can make the AREA statement independent of the screen width. As written, the host spot is always between columns 67 and 80. What happens if the user is running a host session which is 132 columns wide? As it happens, OV will always place the Mail Waiting indicator in columns 67 to 80 regardless of the width of the screen so the AREA statement will still work in that case. If OV expanded to use the full width of the screen, we could write the AREA statement to be independent of the width as follows:

```
SCREEN 1 76 exact * "  A00"  -- OfficeVision main menu
*
* Default buttons will go here...
*
AREA 3 1 13 40
  BUTTON LEFT "{pfkey first}"   Execute PF keys
AREA 14 55 14 80
  BUTTON LEFT "{pfkey}"   Execute PF keys
AREA 0 -14 0 0
  BUTTON LEFT "[pf2]"  Open the mail
```

The last AREA statement now defines a hot spot which is on the last line of the screen, between the 14th column from the end of the screen to the last column of the screen.

Notice that we did not change the other two AREA statements. This is because the PF keys will always appear in the same row/column position no matter what screen size is used. It is safe to use absolute row and column numbers in those AREA statements.

Now to add one more hot spot to the screen. We will allow the user to point at the calendar display to invoke the OV scheduling functions. The OV schedule can be accessed by using the PF1 key, or by entering a command on the command line at the bottom of the screen. We will use the second method. The calendar display on the OV main menu spans from row 5 column 50 to row 13 column 80 (including a bit of margin on both sides). We add another AREA and BUTTON statement to the file:

```
SCREEN 1 76 exact * "  A00"  -- OfficeVision main menu
*
* Default buttons will go here...
*
AREA 3 1 13 40
  BUTTON LEFT "{pfkey first}"   Execute PF keys
AREA 14 55 14 80
  BUTTON LEFT "{pfkey}"   Execute PF keys
AREA 0 -14 0 0
  BUTTON LEFT "[pf2]"  Open the mail
AREA 5 50 13 80
  BUTTON LEFT "[home]appointm[enter]" Calendar functions
```

When the user clicks with the LEFT button anywhere on the calendar, the host cursor will be positioned at the first input field ([home] keyword), the characters `appointm` will be typed on the host screen, and finally an ENTER key will be sent to the host.

Now that we have defined the hot spots for this menu, it is time to consider: what happens if the user clicks the LEFT button outside all of them? For example, what happens if the user clicks with the LEFT somewhere in row 2 of the screen? We have not defined any hot spot in that position. This is where the default button definitions come into play. We left a space for them in the button definition file; they appear after the SCREEN statement and before the first AREA statement. Suppose that if the user clicks the LEFT button outside all the hot spots a pop-up menu will be displayed. We will define the contents of that menu later. For now, all we need to define is the name of the menu. We will pop up a menu named “OVMAIN”:

```
SCREEN 1 76 exact * "  A00"  -- OfficeVision main menu
*
BUTTON LEFT "&{popup ovmain}"  Pop-up OV main menu
*
AREA 3 1 13 40
  BUTTON LEFT "{pfkey first}"   Execute PF keys
AREA 14 55 14 80
  BUTTON LEFT "{pfkey}"   Execute PF keys
AREA 0 -14 0 0
  BUTTON LEFT "[pf2]"  Open the mail
AREA 5 50 13 80
  BUTTON LEFT "[home]appointm[enter]" Calendar functions
```

We have now defined a default LEFT button; if the user clicks the LEFT button outside the hot spots, the menu OVMAIN will pop up. With this button definition, no matter where on the OV main menu the user clicks the LEFT button, CM Mouse will perform some function.

So far we have only looked at the LEFT mouse button. What happens if the user clicks the RIGHT button? We must tell CM Mouse what to do if the user clicks the RIGHT button. Our button usage standards tell us that the RIGHT button should be used to “backup” or “exit” the current menu. In the case of the OV main menu, that means that the RIGHT button should execute a PF12 key to exit OV. The button standards also indicate that the RIGHT button should have the same effect no matter where the mouse is pointing at the time it is pressed. That is, there is not a specific spot on the host screen the user must point to in order to use the RIGHT button. We can easily get the desired effect by adding a BUTTON statement to tell CM Mouse the default function of the RIGHT button:

```
SCREEN 1 76 exact * "  A00"  -- OfficeVision main menu
*
BUTTON RIGHT "[pf12]"             Exit OfficeVision
BUTTON LEFT "&{popup ovmain}"  Pop-up OV main menu
*
AREA 3 1 13 40
  BUTTON LEFT "{pfkey first}"   Execute PF keys
AREA 14 55 14 80
  BUTTON LEFT "{pfkey}"   Execute PF keys
AREA 0 -14 0 0
  BUTTON LEFT "[pf2]"  Open the mail
AREA 5 50 13 80
  BUTTON LEFT "[home]appointm[enter]" Calendar functions
```

Notice we added the BUTTON RIGHT statement before the first AREA statement; thus, it is a default button for this screen. If the user clicks the RIGHT button outside of all the hot spots on this screen, then a PF12 keystroke will be sent to the host. Now what happens if the user clicks the RIGHT button in one of the hot spots? Because the hot spots do not define a function for the RIGHT button (there is no BUTTON RIGHT statement after any of the AREA statements) then CM Mouse uses the default RIGHT button for the screen. Therefore, no matter where the mouse pointer is on the host screen, the RIGHT button will send a PF12 keystroke to the host.

In the same way as we did for the RIGHT button, we can add the combination chord buttons to the definition. If the user presses any chord combination on this screen, we want to scroll to the second OV main menu screen (using PF11):

```
SCREEN 1 76 exact * "  A00"  -- OfficeVision main menu
*
BUTTON LEFT+RIGHT "[pf11]"        Main menu #2
BUTTON RIGHT+LEFT "[pf11]"         Main menu #2
BUTTON RIGHT "[pf12]"              Exit OfficeVision
BUTTON LEFT "&{popup ovmain}"  Pop-up OV main menu
*
AREA 3 1 13 40
  BUTTON LEFT "{pfkey first}"   Execute PF keys
AREA 14 55 14 80
  BUTTON LEFT "{pfkey}"   Execute PF keys
AREA 0 -14 0 0
  BUTTON LEFT "[pf2]"  Open the mail
AREA 5 50 13 80
  BUTTON LEFT "[home]appointm[enter]" Calendar functions
```

We added two BUTTON statements to the list of default buttons for this screen. They both have the same function of sending a PF11 keystroke to the host.

Now let's look at the pop-up menu we are going to use for the default LEFT button. The file name will be OVMAIN.MMM, and it could be written as follows:

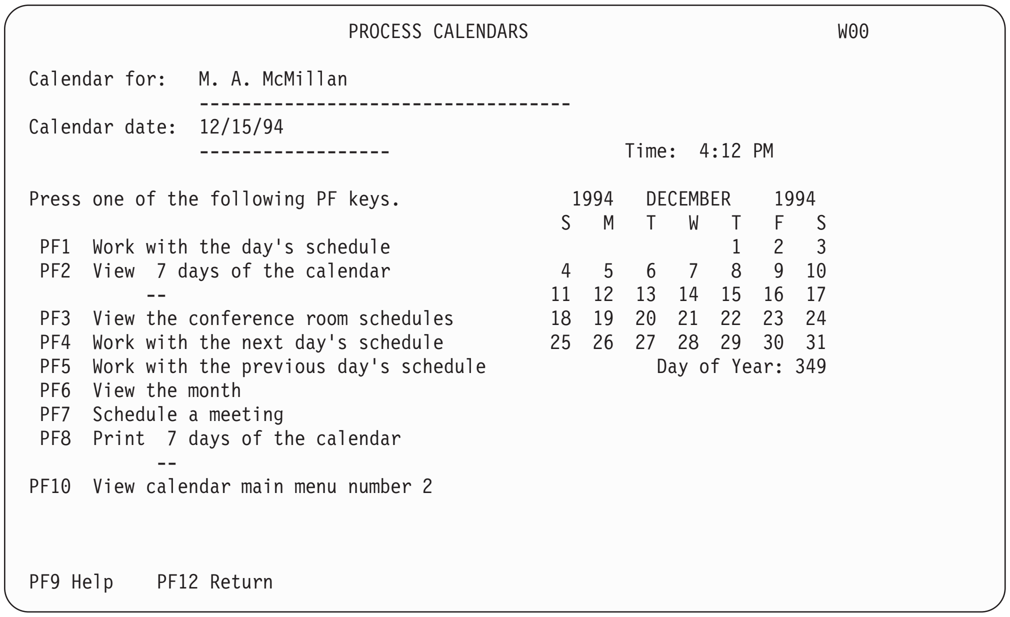
```
TITLE "OV Main Menu"
BAR KEEP
LINE "Open the mail       "[pf2]"
LINE "Calendar            "[pf1]"
LINE "This week's schedule"[pf1][pf2]"
LINE "This months schedule"[pf1][pf6]"
LINE
LINE "FILELIST Utility    "[home]filelist[enter]"
LINE "Spell check a file  "[home]proof &{?File name}[enter]"
LINE
LINE "Edit this list"{editmmm}"
```

This menu duplicates some of the functions available with the point-and-shoot method. It also provides some shortcuts to often-used functions like looking at a week-long schedule. By selecting the last line of the menu the user can add or delete functions from this menu to customize it.

We have now completed the OV main menu. We have written (1) a button definition file defining the screen and hot spots and (2) a pop-up menu.

Calendar Screen Example

Now let's move on to the OV calendar screen. We will use the same ideas of building a SCREEN statement, and then define the hot spots to make it easier to use the scheduling functions. The same button conventions can be used. The OV calendar screen looks something like this:


```
                              PROCESS CALENDARS                            W00

Calendar for:   M. A. McMillan
                -----------------------------------
Calendar date:  12/15/94
                ------------------                    Time:  4:12 PM

Press one of the following PF keys.             1994   DECEMBER    1994
                                               S   M   T   W   T   F   S
 PF1  Work with the day's schedule                             1   2   3
 PF2  View  7 days of the calendar             4   5   6   7   8   9  10
           --                                 11  12  13  14  15  16  17
 PF3  View the conference room schedules      18  19  20  21  22  23  24
 PF4  Work with the next day's schedule       25  26  27  28  29  30  31
 PF5  Work with the previous day's schedule             Day of Year: 349
 PF6  View the month
 PF7  Schedule a meeting
 PF8  Print  7 days of the calendar
            --
PF10  View calendar main menu number 2



PF9 Help    PF12 Return
```

The following example defines the default screen buttons in similar fashion to the preceding main menu example. Here, however, we have an AREA definition allowing us to use the calendar displayed on the screen to do scheduling. If the mouse cursor is pointing to any day within the calendar part of the display, the LEFT button takes you directly to work with that day's schedule. If you select the `Month` title above the calendar, a pop-up menu allows you to select another month and the calendar changes to that month. If you select the year field, a pop-up menu allows you to select another year. If the cursor is not in any of those fields, the LEFT button simply causes a beep.

The SCREEN statement to recognize this menu, as well as the default buttons, would be defined as shown. The SCREEN statement would follow the last AREA statement of the main menu example.

```
   SCREEN 1 76 exact * "  W00" -- OV Calendar menu
     BUTTON LEFT "{beep}"         No function
     BUTTON RIGHT "[pf12]"        Exit calendar menu
     BUTTON LEFT+RIGHT "[pf10]"   2nd calendar menu
     BUTTON RIGHT+LEFT "[pf11]"   2nd calendar menu
```

Now, build the hot spots and define the function of the LEFT button in each one of them. We will start with the “month” title. If the user clicks on the month name shown above the calendar, we will display a pop-up menu of months and allow them to pick another month from the list. That new month will then be entered in the “Calendar date” input field to cause OV to change the calendar display.

```
   AREA 8 58 8 72
     BUTTON LEFT "[home][newline]&{popup
w00month}[enter]"Pick months
```

Note that this assumes a US-style date format with the month as the first two characters of the date field. The W00MONTH.MMM pop-up menu would just provide the 2-character month number:

```
   TITLE "Months"

   LINE "January "01"
```

```
LINE "February"02"
LINE "March   "03"
...
LINE "December"12"
```

Now a similar function for the year. The hot spot area is defined as:

```
AREA 8 54 8 71
  BUTTON LEFT "[home][newline][right][right][right][right][right][right]+
              &{popup w00year}[enter]"Pick year
```

This will pop up the W00YEAR.MMM menu and enter the number selected in the 7th character position of the date field. The W00YEAR menu would be:

```
TITLE "Year"

LINE "1991"91"
LINE "1992"92"
LINE "1993"93"
```

Now if the user clicks in the actual calendar display, CM Mouse goes directly to the schedule for that day. This is done by picking up the word under the mouse cursor (see description of the &{word} keyword on page “&{word delimit|include '<chars>'} or {&word}” on page 76) and entering it in the date field, followed by a PF key:

```
AREA 10 51 14 80 ------------ Calendar area on right side of screen
  BUTTON LEFT "[home][newline][right][right][right]+
               [insert]&{word}[delete][delete][insert][pf1]"
```

Finally, we add two more AREAs to allow the user to click on the PF keys along the left side of the screen, and for the PF keys at the bottom of the screen. Our complete button definition file (including the OV main menu) is:

```
   SCREEN 1 76 exact * "  A00"  ************* OfficeVision main menu **********
   *
   BUTTON LEFT+RIGHT "[pf11]"         Main menu #2
   BUTTON RIGHT+LEFT "[pf11]"         Main menu #2
   BUTTON RIGHT "[pf12]"              Exit OfficeVision
   BUTTON LEFT "&{popup ovmain}"  Popup OV main menu
   *
   AREA 3 1 13 40
     BUTTON LEFT "{pfkey first}"   Execute PF keys
   AREA 14 55 14 80
     BUTTON LEFT "{pfkey}"   Execute PF keys
   AREA 0 -14 0 0
     BUTTON LEFT "[pf2]"  Open the mail
   AREA 5 50 13 80
     BUTTON LEFT "[home]appointm[enter]" Calendar functions

   SCREEN 1 76 exact * "  W00" ****************** OV Calendar menu ************
     BUTTON LEFT "{beep}"         No function
     BUTTON RIGHT "[pf12]"        Exit calendar menu
     BUTTON LEFT+RIGHT "[pf10]"   2nd calendar menu
     BUTTON RIGHT+LEFT "[pf10]"   2nd calendar menu
   AREA 8 58 8 72
     BUTTON LEFT "[home][newline]&{popup
w00month}[enter]"Pick months
   AREA 8 54 8 71
     BUTTON LEFT "[home][newline][right][right][right][right][right][right]+
                 &{popup w00year}[enter]"Pick year
   AREA 10 51 14 80
     BUTTON LEFT "[home][newline][right][right][right]+
                  [insert]&{word}[delete][delete][insert][pf1]"
   AREA 10 1 20 50
     BUTTON LEFT "{pfkey first}"Execute PF key
   AREA 0 1 0 80
     BUTTON LEFT "{pfkey}"Execute PF key
```

RDRLIST Example

The following is a simplified version of the RDRLIST sample files that are supplied with CM Mouse. RDRLIST provides a good example of a “list processing” type of host application in which a list of items is displayed against which commands can be entered.

The VM RDRLIST application menu appears similar to the following screen, but note that many installations and users tailor the display to their own tastes.

```
 MCMILLAN RDRLIST  A0  V 164  Trunc=164 Size=71 Line=1 Col=1 Alt=0
Cmd   Filename Filetype Class User  at Node     Hold  Records  Date      Time
      BUZP     RALVM17  PUN A BUZ      HSTVM17  NONE       56 12/05    09:40:22
      MDQSERV  Note     PUN A MDQSVR   HSTVM17  NONE       56 12/05    14:41:48
      GLEDDEN  RALVM12  PUN A GLEDDEN  HSTVM12  NONE       84 12/05    16:27:24
      BONN     NOTE     PUN A BONN     HSTVM1   NONE       22 12/05    17:45:06
      SCHAF    NOTE     PUN A SCHAF    HSTVM12  NONE       19 12/05    18:21:09
      SCHAF    NOTE     PUN A SCHAF    HSTVM12  NONE        9 12/05    18:22:20
      MANHARP  HSTVM19  PUN A MANHARP  HSTVM19  NONE       17 12/06    08:58:44
      94340    SWP0026  PUN A ZW310    HSTVM10  NONE       60 12/06    10:10:25
      XR06200  AVAIL    PUN B TOWTRUCK HSTVM17  NONE      169 12/06    11:40:49
      DOC211   ZIPBIN   PUN B TOWTRUCK HSTVM17  NONE     1172 12/06    11:40:50
      CASANNA  MSNVM1   PUN A CASANNA  HSTVM1   NONE       60 12/06    14:58:16
      94340    TG30444  PUN A INFORMU  HSTVM02  NONE      146 12/06    15:09:33
      Acknowl  edgment  PUN A JL17598  HSTVM4   NONE        2 12/06    20:59:26
      IBMPC    DONE     PUN B IBMPC    HSTVMV   NONE        6 12/07    07:53:38
      IBMPC    DONE     PUN B IBMPC    HSTVMV   NONE        6 12/07    07:57:13
      Acknowl  edgment  PUN A GERSTLE  HSTVMK   NONE        2 12/07    08:49:43
      IBMPC    DONE     PUN B IBMPC    HSTVMV   NONE        6 12/07    09:18:15
1= Help      2= Refresh  3= Quit      4= Sort(type) 5= Sort(date) 6= Sort(user)
7= Backward  8= Forward  9= Receive  10=           11= Peek      12= Cursor

====>
                                                            X E D I T  1 File
```

Figure 17. Sample VM RDRLIST Application Menu

In this example, we use different column ranges in the list for different functions. For example, if the mouse cursor is in the `Cmd` field (columns 1 to 5), then we use the LEFT button to pop up a list of commands that can be executed against the file the mouse cursor is next to. If the cursor is immediately next to the file name (column 6), then a “repeat” marker is put in the command field. If the mouse cursor is in the `Filename` or `Filetype` columns, the file is PEEKed. The idea here is that the user should be able to just point at a file and click the LEFT button to perform some default action (the most commonly used action). This saves users the trouble of popping up a menu of commands when they most often do the same action on each file in the list. In the `Hold` field (columns 49 to 52), the LEFT button is used to DISCARD the file. In any other column, the LEFT button is used to move the host cursor to the beginning of that line, and the mouse can be used on one of the PF keys at the bottom of the screen. This provides, in effect, the ability to pick a file and then pick the command (PF key) to run on it. This is for example purposes only; you may choose to define the buttons or AREAs differently according to your own tastes and needs. In the Command line area at the bottom of the screen, a command menu appears.

The sample RDRLIST configuration supplied with CM Mouse defines the columns somewhat differently, and allows the user to easily tailor the file to accommodate customized PF keys.

As in the OfficeVision example, the RIGHT key exits RDRLIST, and the combination keys are used for scrolling. The button definition for RDRLIST to perform the functions outlined earlier is shown in the example. Note the use of offset (negative) values for AREAs whose boundaries change depending on screen size.

```
screen 1 10 exact * " RDRLIST " -- VM RDRLIST screen
button left "{beep}"     No function
button right "[pf3]"     Exit RDRLIST
button left+right "[pf8]"Scroll down
button right+left "[pf7]"Scroll up

area 3 1 -5 5 ****************************** Command columns
  button left "&{popup rdrfiles}"Popup command list

area 3 6 -5 6 ****************************** Adjacent column
  button left "{seek}[backtab]="Repeat marker

area 3 49 -5 52 **************************** Hold field
  button left "{seek}[pf6]"Discard file

area 3 1 -5 80 ***************************** Anywhere else in file area
  button left "{seek}[backtab]"Move cursor

area -4 1 -3 80 **************************** PF key list area
  button left "{pfkey}"Execute PF key

area -2 1 0 0 ****************************** Command area
  button left "&{popup rdrcmds}"Popup VM commands
```

The menu file RDRFILES.MMM might look like:

```
  title " RDRLIST File Disposition "
  line "Look at the file        "{seek}[pf2]
  line "Remove from the reader   "{seek}[pf6]
  line "Put on my A-disk/notebook"{seek}[pf10]
  line "Run repeated commands    "{seek}[pf10]
  line "Put in XXYY notebook     "{seek}receive / (notebook xxyy[enter]
  line "Put in another notebook  "{seek}receive / (notebook [eraseeof]
  line "Incomplete RECEIVE cmd   "{seek}receive / [eraseeof]
  line
  line "Other RDRLIST commands   "&{popup rdrcmds}"
```

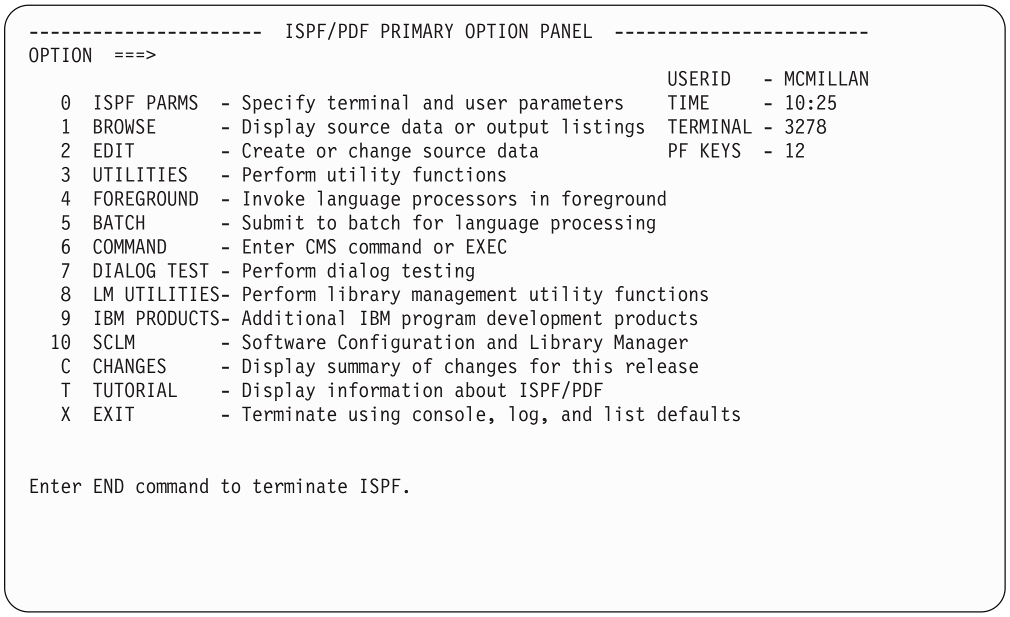
The RDRCMDS.MMM menu file might look like:

```
  title " RDRLIST Commands "
  line "Rebuld the list"[pf11]
  line "Sort by type   "[pf4]
  line "Sort by date   "[pf5]
  line "Quit RDRLIST   "[pf3]
  line
  line "File disposition"&{popup rdrfiles}
```

As with the OV examples, this is just a sample of what types of things can be done. You would probably want to customize the menu files to do the things that you do most often. You might also want to change how the buttons are defined in various areas of the screen.

ISPF Example

The ISPF main menu appears similar to the following screen:



```
----------------------  ISPF/PDF PRIMARY OPTION PANEL  ------------------------
OPTION  ===>
                                                            USERID   - MCMILLAN
   0  ISPF PARMS  - Specify terminal and user parameters    TIME     - 10:25
   1  BROWSE      - Display source data or output listings  TERMINAL - 3278
   2  EDIT        - Create or change source data            PF KEYS  - 12
   3  UTILITIES   - Perform utility functions
   4  FOREGROUND  - Invoke language processors in foreground
   5  BATCH       - Submit to batch for language processing
   6  COMMAND     - Enter CMS command or EXEC
   7  DIALOG TEST - Perform dialog testing
   8  LM UTILITIES- Perform library management utility functions
   9  IBM PRODUCTS- Additional IBM program development products
  10  SCLM        - Software Configuration and Library Manager
   C  CHANGES     - Display summary of changes for this release
   T  TUTORIAL    - Display information about ISPF/PDF
   X  EXIT        - Terminate using console, log, and list defaults


Enter END command to terminate ISPF.
```

Figure 18. Sample ISPF Main Menu

A simple button definition file definition would work well for this screen:

```
screen 1 26 exact * "ISPF/PDF PRIMARY OPTION PANEL"
button left "&{popup ispfcmds}
button right "[pf3]
area 4 1 19 80 ---------------- List of options
  button left "{word first}[enter]
area 21 1 21 40 --------------- END message
  button left "=x[enter]"
```

Note that in this case we used the title as the key to recognizing the screen. Like the OfficeVision samples given earlier, a number of additional SCREEN definitions would be required to use the mouse throughout the whole ISPF system. The preceding is a sample for the main menu only. The ISPFCMDS.MMM file might be used to take you directly to the ISPF panels that you use most often, such as:

```
title "My ISPF Options"
line "Browse my net log"1[enter][newline][newline][newline]+
                        [newline][newline]mcmillan netlog[enter]
line "List files"3.4[enter][newline][newline]+
                 [newline][newlinw]* * A[enter]"
line
line "Exit ISPF"=x[enter]
```

Text Editors

CM Mouse can be useful for host text editing in several ways:

1. It is much easier to move the mouse cursor quickly across the screen, particularly in diagonal directions, than to use the keyboard arrow keys. If one of the mouse buttons is set to {seek}, pressing that button moves the host cursor to the mouse cursor position.
2. Mouse buttons can be programmed to perform some common editing keystrokes such as scrolling, inserting lines, and deleting lines.
3. Mouse buttons can be programmed to do very specific and complex editing functions. This is particularly useful when you want to perform an editing function at various places in the file, but you cannot use a global change

command (perhaps you do not want to change all occurrences or you cannot adequately specify the correct locations). The CM Mouse/REXX interface can be useful in this case. For example, suppose you wanted to change certain words in the file to upper case. You might not want to change all occurrences of a word, but just some. You could define a mouse button which when you pointed to a word and clicked, that word would convert to upper case:

```
button left "{seek}[wordleft]{&rexx +
                xword = '{&word at {&hrow} {&hcol}}';+
                xword = translate(xword);+
                exit xword;}"
```

By defining other buttons as scroll keys, you can move quickly through a file making changes only where desired.

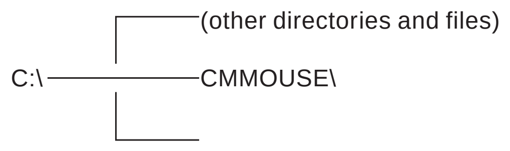
4. Pop-up menus can be used to select or change various editing options.
5. The Cut and Paste functions can be useful for copying portions of text from one place to another. This is useful for moving rectangular portions of text (as opposed to whole lines) since most host text editors are not well suited to such operations.

Chapter 15. Tips and Techniques

Typical Installation

The most common and simplest way to install and use CM Mouse is to put all the files in a CMMOUSE subdirectory of the default drive's root directory (usually C:\). The CM Mouse defaults have been designed assuming that all CM Mouse button definition and menu files exist in that subdirectory. Using this technique prevents cluttering up the root directory with button and menu files. It also improves performance because CM Mouse only needs to search through the C:\CMMOUSE\ subdirectory, which has have very few files compared to the root directory.

If you install all the .BDF and .MMM files in the CM Mouse package into the directory named C:\CMMOUSE\, you can simply take the default values for the *bdf*, *bdfdir*, and *mmmdir* parameters when you start CM Mouse. Your directory structure will look like this:



If you need to specify any parameters when you start CM Mouse, such as *mmmdir* or *bdfdir*, it is much easier if they are specified using OS/2 environment variables. By placing SET statements in your system initialization file, you do not have to type in all the parameters when you start CM Mouse. For example, you might put the following lines into your CONFIG.SYS (OS/2) or AUTOEXEC.BAT (Windows) file:

```
SET CMMOUSE.MMMDIR=C:\CMMOUSE\MENUS\
SET CMMOUSE.BDFDIR=C:\CMMOUSE\BUTTONS\
SET CMMOUSE.INITSESS=b
```

See “Environment (SET) Variables” on page 7 for details.

Network Installation

If CM Mouse is to be used by a large number of users on a network, there are several advantages to installing the program and BDF/MMM files on a LAN server. This allows them to be updated in a single place if a new release level needs to be installed and can save disk space on each user's system.

Note!

You should ensure that installation of this program on a network server does not violate any license restrictions under which the program was purchased.

CM Mouse has been designed so that users in a LAN network environment can customize their own set of BDF and MMM files without affecting other users on the network. It is assumed that there is a large number of BDF and MMM files to which all users require access. Some users will want to modify a small subset of those

BDF and MMM files. For example, some users might change the order of menu items in the MMM file to move the options they use most to the top of the menu.

The setup of CM Mouse in this environment involves installing the BDF and MMM files in a directory on the LAN server which is available to all users. For example, if the server disk appears to each client system on the LAN as the R drive, then you might install all the BDF and MMM files under a subdirectory R:\CMMOUSE. The EXE and DLL files might reside in some other directory on the server that is in the OS/2 PATH and LIBPATH lists.

On each client system, a subdirectory should be created to hold any changed BDF or MMM files. This might be C:\CMMOUSE, for example. In each client's CONFIG.SYS file, the following lines should be added:

```
SET CMMOUSE.BDFDIR=C:\CMMOUSE;R:\CMMOUSE
SET CMMOUSE.MMMDIR=C:\CMMOUSE;R:\CMMOUSE
```

This causes CM Mouse to first look on the client system C drive for BDF and MMM files and, if they are not found there, to search the server disk for the R drive.

If you do nothing, then all BDF and MMM files are fetched from the server. However, if you want to change a pop-up menu, you can copy that MMM file from the server to your local system and edit the file. This is done automatically by CM Mouse when you select the **Edit this menu** option from the pop-up's system menu (see "Pop-up System Menus" on page 13). The menu file is copied from wherever it was fetched to the first directory in the MMMDIR list. In this example, the file is copied from the server to the your C drive. Any changes you now make to that file affects only that client system and not any other users of the server.

There is no facility in CM Mouse to automatically copy BDF files from one directory to another. A user who wants to change a BDF file in this environment must manually copy the BDF file from the server to another directory earlier in the BDFDIR list.

Nesting Pop-up Menus

CM Mouse does not really support nested pop-up menus, but because one pop-up can invoke another with the &{popup ...} substitution word, you can create a similar effect by including items on the menus that invoke other menus. For example, if you have a very long list of options, you can break them into three lists and code them as follows:

The first menu ("menu1of3.mmm") might look like this:

```
title " List (1 of 3) "
line "Item # 1"...
line "Item # 2"...
...
line "Item # 15"...
line "  <<< more >>>"&{popup menu2of3}
```

The second ("menu2of3.mmm"):

```
title " List (2 of 3) "
line "  <<< more >>>"&{popup menu1of3}
line "Item 16"...
line "Item 17"...
...
line "Item 25"...
line "  <<< more >>>"&{popup menu3of3}
```

The third (“menu3of3.mmm”):

```
title " List (3 of 3) "
line "  <<< more >>>"&{popup menu2of3}
line "Item 26"...
line "Item 27"...
...
line "Item 37"...
```

These lists could be made circular by having the first item of the first list invoke menu 3 and the last item of menu 3 invoke menu 1. These lists are not really nested; pressing the RIGHT button to exit a menu does not cause a return to the previous menu. The user must explicitly select the next menu to be shown.

Synchronizing Input with the Host

In some cases, a series of keystrokes being sent to the host will “overrun” it; keystrokes will be lost because they are sent to the host before the host is ready to receive them. In normal circumstances, if the host is not ready to receive keyboard input, the keyboard is locked and keystrokes are queued until the host is ready. In some cases, however, the keyboard may be unlocked for a period of time before the host application is actually ready to receive input.

One example of an overrun condition occurs when an application is started from VM/CMS. The CMS command is read and processing begins, but before the application displays the first menu, the keyboard is unlocked and CMS continues to solicit input. Keystrokes and PF keys entered at that time can be lost to the application or can be executed later from the CMS command stack. Other applications (notably ISPF on MVS) can also unlock the keyboard at inappropriate times, causing keystroke overrun.

For instance, if a button definition is to invoke OfficeVision and go directly to the CALENDAR function from CMS mode, the following sequence will not work:

```
"office[enter][pf1]"
```

After the [enter] key is processed (and before the OV main menu is displayed), the PF1 key is “stacked” by CMS. It is executed as a CMS PF key command when OfficeVision terminates.

In such a case, several approaches are possible. A CM Mouse {pause xx} control word can be imbedded in the command string as follows:

```
"office[enter][pause 2000][pf1]"
```

This has the obvious deficiency that it assumes OV will take less than 2 seconds to put up the main menu. Slow system response can cause this technique to fail. It also introduces a 2-second delay in processing the request, even if the system response time is good. The {vmwait} control word was specifically designed to eliminate this problem on VM. The preceding example works much better as:

```
"office[enter]{vmwait 10}[pf1]"
```

This technique does not introduce any additional delay if the host responds quickly, and does not fail until the host does not respond for more than 10 seconds.

Another way to handle this situation is to use the {search...} control word to stop execution until the menu appears on the host screen. CM Mouse waits for a defined period of time for the given string to appear anywhere on the screen. Note that the amount of time CM Mouse waits can be specified as part of the {search...} keyword.

For example, to invoke OV from the CMS command mode and go directly to the calendar function, the following button definition could be used:

```
"office[enter]{search for 'OfficeVision' at 1 1 1 80 wait 10}[pf1]"
```

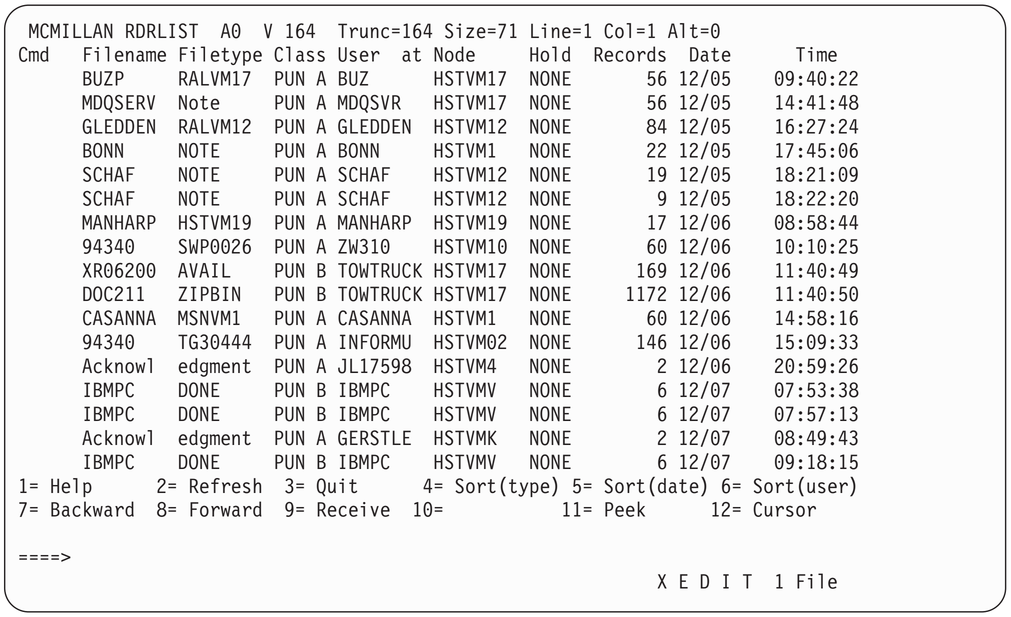
This will enter the command `office` and wait up to 10 seconds for the characters `OfficeVision` to appear anywhere in line 1 of the host screen. If the characters appear within 10 seconds, a PF1 key is sent to the host which invokes the calendar function on the main menu. If the characters do not appear within 10 seconds, the [pf1] keyword is discarded.

In general, the {search...} keyword can be used to synchronize the sending of keystrokes to the host.

Screen Size Independence

The emulation programs support the emulation of several different models of 3270 display terminals. Therefore several different configurations of screen sizes are supported, such as 24 lines by 80 columns, 32 lines by 80 columns, 28 lines by 132 columns, and so on. CM Mouse button definition files must be carefully written to work correctly on all sizes of host screens.

Consider the following host screen:
To allow the user to click on the PF keys as shown on the screen, the following



```
 MCMILLAN RDRLIST  A0  V 164  Trunc=164 Size=71 Line=1 Col=1 Alt=0
Cmd   Filename Filetype Class User  at Node     Hold  Records  Date      Time
      BUZP     RALVM17  PUN A BUZ      HSTVM17  NONE       56 12/05    09:40:22
      MDQSERV  Note     PUN A MDQSVR   HSTVM17  NONE       56 12/05    14:41:48
      GLEDDEN  RALVM12  PUN A GLEDDEN  HSTVM12  NONE       84 12/05    16:27:24
      BONN     NOTE     PUN A BONN     HSTVM1   NONE       22 12/05    17:45:06
      SCHAF    NOTE     PUN A SCHAF    HSTVM12  NONE       19 12/05    18:21:09
      SCHAF    NOTE     PUN A SCHAF    HSTVM12  NONE        9 12/05    18:22:20
      MANHARP  HSTVM19  PUN A MANHARP  HSTVM19  NONE       17 12/06    08:58:44
      94340    SWP0026  PUN A ZW310    HSTVM10  NONE       60 12/06    10:10:25
      XR06200  AVAIL    PUN B TOWTRUCK HSTVM17  NONE      169 12/06    11:40:49
      DOC211   ZIPBIN   PUN B TOWTRUCK HSTVM17  NONE     1172 12/06    11:40:50
      CASANNA  MSNVM1   PUN A CASANNA  HSTVM1   NONE       60 12/06    14:58:16
      94340    TG30444  PUN A INFORMU  HSTVM02  NONE      146 12/06    15:09:33
      Acknowl  edgment  PUN A JL17598  HSTVM4   NONE        2 12/06    20:59:26
      IBMPC    DONE     PUN B IBMPC    HSTVMV   NONE        6 12/07    07:53:38
      IBMPC    DONE     PUN B IBMPC    HSTVMV   NONE        6 12/07    07:57:13
      Acknowl  edgment  PUN A GERSTLE  HSTVMK   NONE        2 12/07    08:49:43
      IBMPC    DONE     PUN B IBMPC    HSTVMV   NONE        6 12/07    09:18:15
1= Help      2= Refresh  3= Quit      4= Sort(type) 5= Sort(date) 6= Sort(user)
7= Backward  8= Forward  9= Receive  10=           11= Peek      12= Cursor

====>
                                                            X E D I T  1 File
```

Figure 19. Sample Host RDRLIST Screen

BDF fragment could be written:

```
AREA 20 1 21 80
BUTTON LEFT "{pfkey}"
```

Suppose, however, that the user was running this host application with a screen size of 32 lines by 80 columns. In this case, the application would display the PF keys on lines 28 and 29. The above AREA statement would incorrectly create a hot spot on lines 20 and 21. The problem comes from the specification of the row and column coordinates of the AREA as *absolute* coordinates. That is, the coordinates are in terms of the upper-left corner of the screen and remain in the same position no matter what screen size is used.

To avoid this problem of being dependent on screen size, CM Mouse allows you to specify an AREA in *relative* coordinates. Relative coordinates are specified as offsets from the last row of the screen (no matter what row number it happens to be). A relative coordinate is a negative or zero value. A value of zero is the last row of the screen, -1 is the row immediately above it, -2 is the row above -1, and so on. Thus the above BDF fragment could be written:

```
AREA -4 1 -3 80
BUTTON LEFT "{pfkey}"
```

This tells CM Mouse to define a hot spot which starts at the 5th row up from the bottom of the screen, column 1, and goes to the 4th row up from the bottom, column 80. Note that "-4" specifies the 5th row from the bottom because the bottom row is zero. The above AREA statement will correctly define the hot spot no matter how many rows are in the host screen.

In much the same way that row numbers can be specified as relative values, column numbers can be specified as zero and negative numbers. In the example above, if the user was using a host screen of 28 rows by 132 columns the hot spot would only cover the first 80 columns of the screen. To be completely correct, the above fragment should be written:

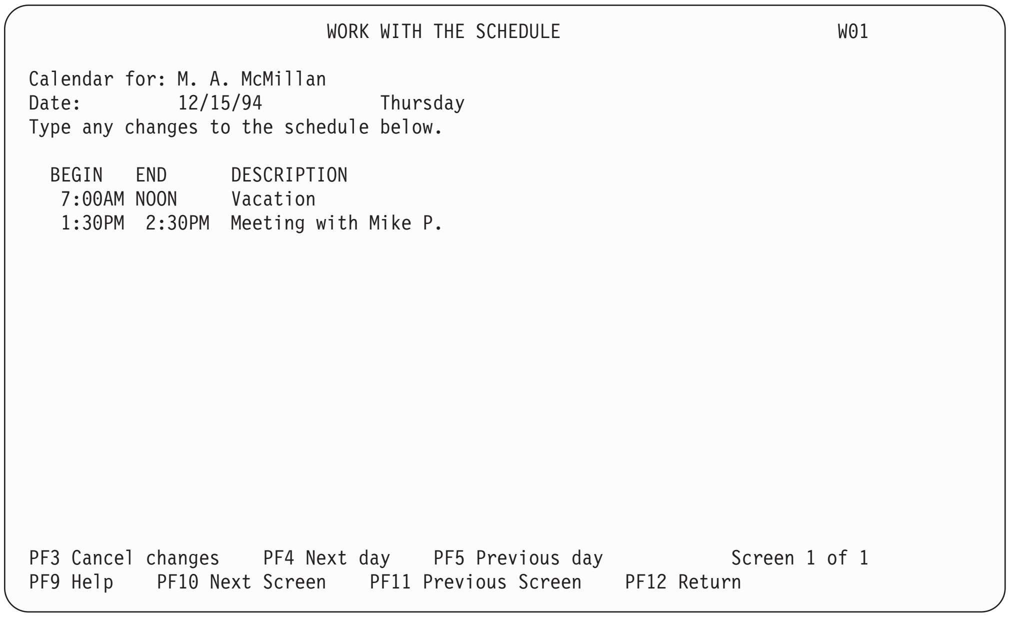
```
AREA -4 1 -3 0
BUTTON LEFT "{pfkey}"
```

This would define a hot spot which covers the 4th and 5th row from the bottom of the screen, from the left edge to the right edge. This AREA statement will be correct for any size host screen.

It is important to consider different screen sizes when defining hot spots. Consider different screen sizes both in length and width and write AREA statements using *relative* coordinates when appropriate.

Cursor Positioning

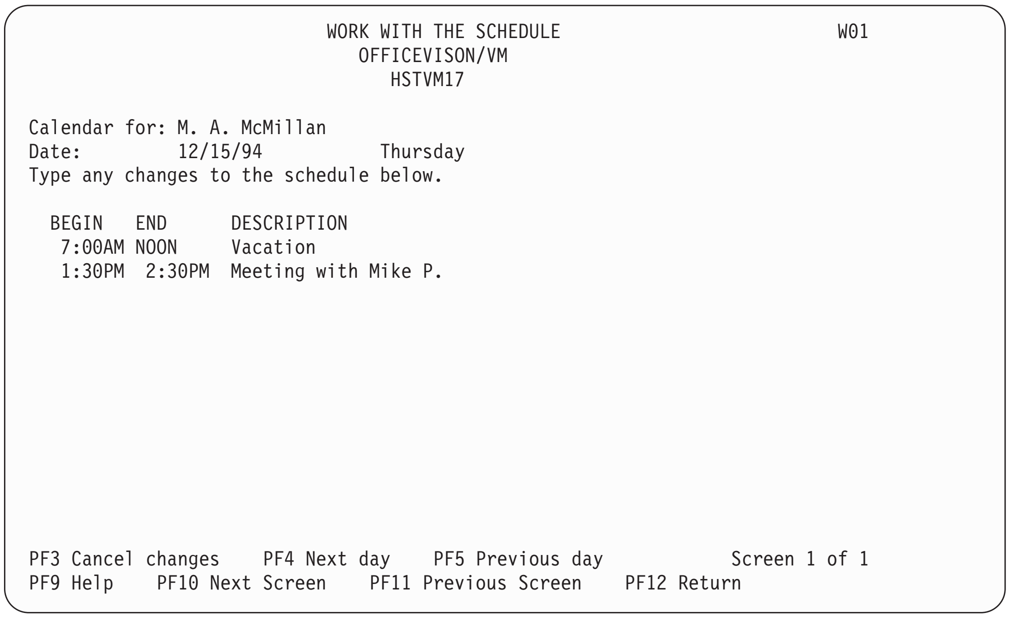
To make button definitions that are tolerant of changes to the host screen layout, it is useful to define cursor positions in the most general way possible. When the host cursor needs to be positioned at a particular input field of the screen, it is best to use the various field-tab keys to place the cursor rather than position the cursor at a specific row and column. For example, consider the host screen:


```
                            WORK WITH THE SCHEDULE                          W01

Calendar for: M. A. McMillan
Date:         12/15/94          Thursday
Type any changes to the schedule below.

  BEGIN   END      DESCRIPTION
   7:00AM NOON     Vacation
   1:30PM  2:30PM  Meeting with Mike P.















PF3 Cancel changes    PF4 Next day    PF5 Previous day            Screen 1 of 1
PF9 Help    PF10 Next Screen    PF11 Previous Screen    PF12 Return
```

If you wanted to write a button definition which would (for example) type the current date into the date field, you could write:

```
BUTTON LEFT "{rowcol 4 16}&{month}/&{day}/&{year}"
```

This would position the host cursor at row 4 column 16 and then type the current month, day, and year. What happens, however, if small changes are made to the host application? For example, suppose the above host application screen was modified to add two additional title lines:
Now the button definition does not work because the input field has moved from


```
                            WORK WITH THE SCHEDULE                          W01
                               OFFICEVISON/VM
                                  HSTVM17

Calendar for: M. A. McMillan
Date:         12/15/94          Thursday
Type any changes to the schedule below.

  BEGIN   END      DESCRIPTION
   7:00AM NOON     Vacation
   1:30PM  2:30PM  Meeting with Mike P.













PF3 Cancel changes    PF4 Next day    PF5 Previous day            Screen 1 of 1
PF9 Help    PF10 Next Screen    PF11 Previous Screen    PF12 Return
```

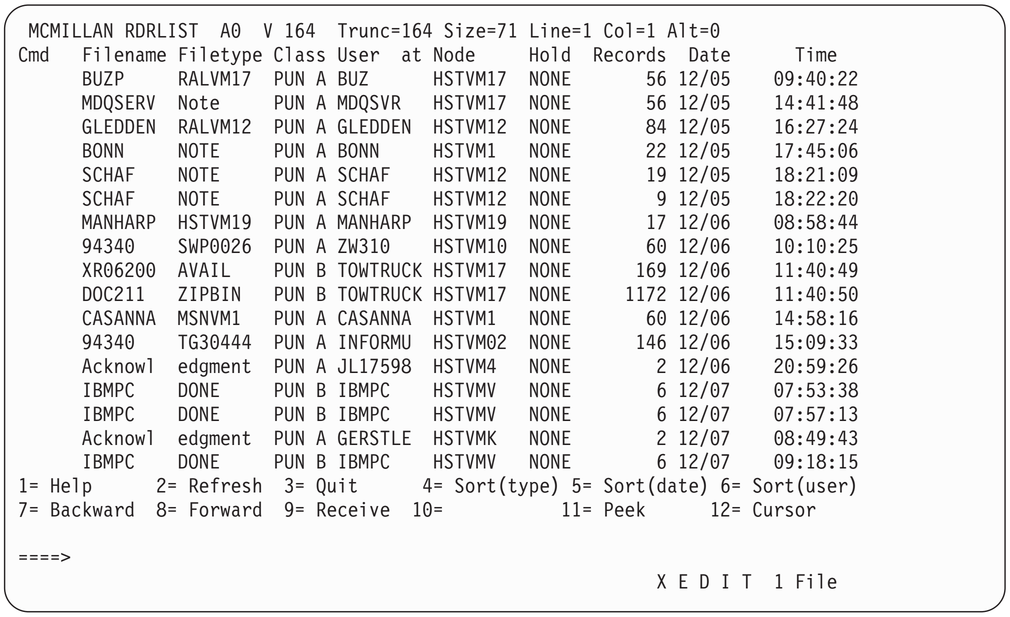
line 4 to line 6. A better button definition would be:

```
BUTTON LEFT "[home][tab]&{month}/&{day}/&{year}"
```

This would position the host cursor to the first input field of the screen (no matter what line it was on), then skip to the next input field and type the current date. This button definition continues to work even with the modified host screen as long as the date field is the second input field. The button definition is more tolerant of host screen changes than the first example.

It is not possible to write button definitions which tolerate any arbitrary change to the host screen. For example, if the host screen above were modified to have the date field first followed by the name field, our button definition would not work correctly. However, by writing button definitions with as much tolerance as possible, button definitions can continue to work even if minor changes are made to the host screens.

Another example of good cursor positioning technique can be shown with the following host screen:
Note that the command line is at the bottom of the screen. To position the host



```
 MCMILLAN RDRLIST  A0  V 164  Trunc=164 Size=71 Line=1 Col=1 Alt=0
Cmd   Filename Filetype Class User  at Node     Hold  Records  Date      Time
      BUZP     RALVM17  PUN A BUZ      HSTVM17  NONE       56 12/05    09:40:22
      MDQSERV  Note     PUN A MDQSVR   HSTVM17  NONE       56 12/05    14:41:48
      GLEDDEN  RALVM12  PUN A GLEDDEN  HSTVM12  NONE       84 12/05    16:27:24
      BONN     NOTE     PUN A BONN     HSTVM1   NONE       22 12/05    17:45:06
      SCHAF    NOTE     PUN A SCHAF    HSTVM12  NONE       19 12/05    18:21:09
      SCHAF    NOTE     PUN A SCHAF    HSTVM12  NONE        9 12/05    18:22:20
      MANHARP  HSTVM19  PUN A MANHARP  HSTVM19  NONE       17 12/06    08:58:44
      94340    SWP0026  PUN A ZW310    HSTVM10  NONE       60 12/06    10:10:25
      XR06200  AVAIL    PUN B TOWTRUCK HSTVM17  NONE      169 12/06    11:40:49
      DOC211   ZIPBIN   PUN B TOWTRUCK HSTVM17  NONE     1172 12/06    11:40:50
      CASANNA  MSNVM1   PUN A CASANNA  HSTVM1   NONE       60 12/06    14:58:16
      94340    TG30444  PUN A INFORMU  HSTVM02  NONE      146 12/06    15:09:33
      Acknowl  edgment  PUN A JL17598  HSTVM4   NONE        2 12/06    20:59:26
      IBMPC    DONE     PUN B IBMPC    HSTVMV   NONE        6 12/07    07:53:38
      IBMPC    DONE     PUN B IBMPC    HSTVMV   NONE        6 12/07    07:57:13
      Acknowl  edgment  PUN A GERSTLE  HSTVMK   NONE        2 12/07    08:49:43
      IBMPC    DONE     PUN B IBMPC    HSTVMV   NONE        6 12/07    09:18:15
1= Help      2= Refresh  3= Quit      4= Sort(type) 5= Sort(date) 6= Sort(user)
7= Backward  8= Forward  9= Receive  10=           11= Peek      12= Cursor

====>
                                                            X E D I T  1 File
```

cursor on the command line, you could write:

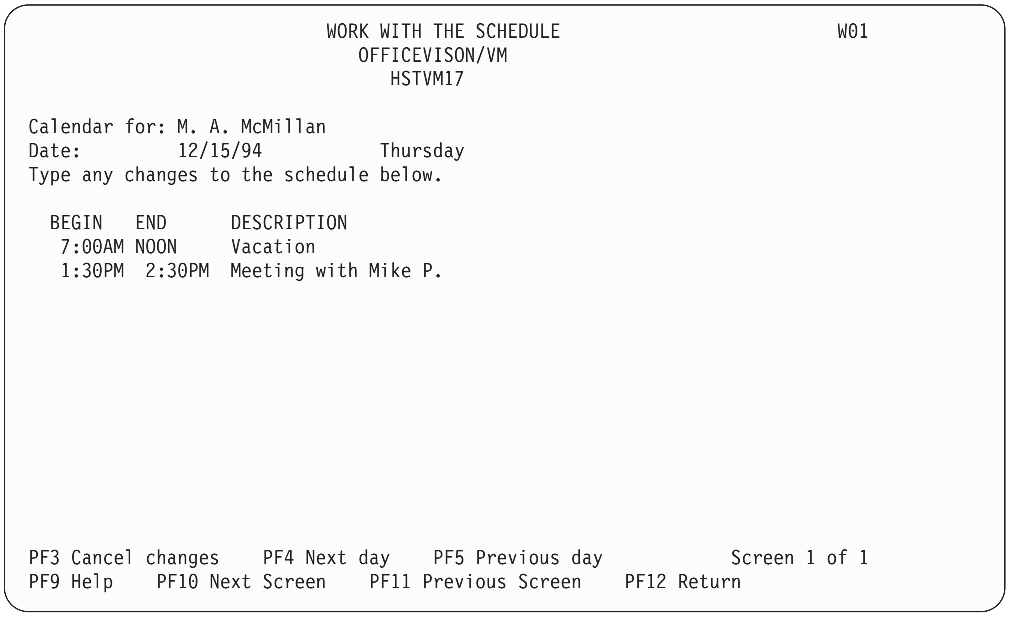
```
BUTTON LEFT "{rowcol -1 7}mycmd[enter]"
```

This would work on any size host screen since the cursor position is given in relative coordinates. However, this function can be better written:

```
BUTTON LEFT "[home][backtab]mycmd[enter]"
```

This would position the cursor at the last input field of the screen and will work even if the command line is moved up or down a row or two. The "[home][backtab]" technique should be used anytime the host cursor needs to be positioned at the last input field of the screen.

As another example of cursor positioning techniques, consider:


```
                              WORK WITH THE SCHEDULE                          W01
                                 OFFICEVISON/VM
                                    HSTVM17

Calendar for: M. A. McMillan
Date:         12/15/94           Thursday
Type any changes to the schedule below.

  BEGIN   END      DESCRIPTION
   7:00AM NOON     Vacation
   1:30PM  2:30PM  Meeting with Mike P.









PF3 Cancel changes    PF4 Next day    PF5 Previous day            Screen 1 of 1
PF9 Help    PF10 Next Screen    PF11 Previous Screen    PF12 Return
```

Suppose that we define a hotspot that covers the “BEGIN” column of input fields. When the user clicks in that field, the string “8:00AM” is entered in the BEGIN column, and “9:00AM” is entered in the END column:

```
AREA 10 3 -2 10
BUTTON LEFT "{seekcol 3}8:00am[tab]9:00am"
```

Notice that the cursor is moved to the 3rd column of the row of the mouse cursor because that is where the input field starts. However, if the host application is changed slightly so that (for example) the input field started in column 5, then this button definition would no longer work. This definition could be better written:

```
AREA 10 3 -2 10
BUTTON LEFT "{seekcol 1}[tab]8:00am[tab]9:00am"
```

Written this way, the host cursor is first moved to column 1 and then tabbed to the first input field of the line. This button definition would work no matter in which column the first input field started.

Performance Tips

CM Mouse performance can potentially be degraded by having a very large number of SCREEN statements in the button definition files, particularly for slower PC hardware. Each time a mouse button is pressed, CM Mouse must scan all the incore SCREEN definitions to find a match. Depending on how the SCREEN definitions are written, this can involve a fairly large amount of processing (each definition must be checked against the current host screen). On slower systems (such as ATs), SCREENs appearing later in the BDF sequence may take noticeably longer to recognize.

Following are some tips for optimizing CM Mouse performance:

1. Place the button definitions for the most frequently-used applications first in the button definition file. SCREENs are searched for a match in order of appearance in the button definition file. The first SCREENs in the file are found more quickly than those at the end of the file if the file contains a large number of SCREENs. Note that INCLUDEd files are treated as if they are part of the file which includes them.

2. Whenever possible, code SCREEN statements for a specific host session (a, b, c...) or for a specific session type (5250/3270). The first check CM Mouse makes is to see if the SCREEN definition is for the current host session. If the session is coded as “*”, then more checking must be done to see if the definition matches.
3. Whenever possible, specify an EXACT or LINE search instead of a SCREEN search. EXACT and LINE searches take the same amount of time, but a SCREEN search can take significantly longer.
4. As with SCREENs, place AREA definitions within a SCREEN in the order of most frequent use, if possible. (Note that when AREAs overlap, changing the order can affect how they work.)

Common Problems

Following are some common problems which could be encountered in the installation and use of CM Mouse.

1. *Screens do not appear to be recognized (for example, the LEFT button generates a “beep,” and pop-up menus do not appear).*

   There are a number of possible causes for this:

   - Be sure the button definition file (BDF) which contains the SCREEN statement is being read when CM Mouse starts up. Be sure your main BDF file (usually c:\cmmouse\cmmouse.bdf) has the proper INCLUDE statements and that they are not commented out.
   - Use the Screen Map (Map Trace window) to determine if the wrong SCREEN or AREA is being used.
   - Be sure the SCREEN statement you are trying to use is exactly correct, including the host session ID. The safest way to code SCREEN statements is to use a session ID of “*”.
2. *After you stop and restart a host session, screens are not recognized.*

   You must select the “Reset Host” button on the CM Mouse control panel after stopping or starting any host session.
3. *Changing a button definition file (BDF) does not appear to have any effect on the program.*

   You must select the **Read BDFs** button on the CM Mouse control panel after changing any BDF files. CM Mouse reads BDF files only when it is first initialized and when the **Read BDFs** button is selected.

Chapter 16. Cross Platform Compatibility

This section describes areas of incompatibility or differences between CM Mouse running in the various environments.

By using only button definition keywords which are supported in all environments, BDF and MMM files can be moved from one environment to the other without modification.

CM Mouse has been designed so that button definition and pop-up menu files can be written which can be moved from one environment to another and used without modification. In general, the OS/2 version of CM Mouse supports the most CM Mouse statements and keywords. The Windows version supports a subset of the OS/2 function (and a few keywords unique to Windows). If compatibility between environments is important, then limiting the use of CM Mouse statements and keywords to those supported in the environments of interest will make the button definition and pop-up menu files portable between those environments. The keyword table in “Appendix A. CM Mouse Keyword Reference” on page 147 indicates which keywords are supported in which environments.

Wherever it is reasonable to do so, statements or keywords which are not supported in a particular environment are simply ignored, but cause no error. Where it could cause incorrect operation, unsupported statements or keywords cause an error (for example, attempting to run a REXX keyword in the Windows version of CM Mouse causes an error). This allows flexibility in moving button definition and menu files between environments, even if all the features they use may not be supported.

The following table summarizes the major areas of compatibility.

Function	**OS/2**	**Windows**
Basic host keystrokes [PF1] [enter] etc.	✓	✓
Basic CM Mouse keywords and substitutions	✓	✓
Print screen to printer or file	✓	✓
CM Mouse control panel for customizing	✓	✓
Customizable system menu (CMMOUSE.MMM)	✓	✓
Clipboard functions and keywords	✓	✓
Window manipulation keyword {win...}	✓	✓
Double-click button support	✓	✓
REXX language interface	✓	
Screen Map facility	✓	
Trace facility		✓
5250 terminal support	✓	✓

Chapter 17. Limitations

This chapter describes some of the limitations of CM Mouse. There may exist other limitations which are not documented here.

String Lengths

Unless stated otherwise in the following table, strings in CM Mouse are limited to 4000 characters in length (255 for DOS). This limit includes the maximum length of a BDF or MMM statement after line continuation. Button definition strings cannot exceed this limit before, during, or after substitutions.

String	Max Length (OS2, WIN)
Input line (after continuation)	4000
CM Mouse variable name	4000
CM Mouse variable value	4000
Button definition string	4000
File names (including full path qualifier)	255
Pop-up menu title	250
Pop-up menu text (length of ’text’ part of LINE statement)	250

Program Limitations

The following table lists some other program limitations.

Description	Limit (OS/2)	Limit (Windows)
Maximum number of lines in a single BDF file	None	32,000
Maximum number of INCLUDE statements in all BDF files	None	32,000
Maximum number of lines in all BDF files	None	None
Maximum nesting level of BDF INCLUDE files	4	4
Maximum number of SCREEN statements	None	32,000
Maximum number of AREA statements	None	32,000
Maximum number of BUTTON statements	None	None
Maximum number of concurrent host sessions	26	26
Maximum width (columns) on a host session	255	255
Maximum length (rows) on a host session	None	None

Miscellaneous Limitations

- CM Mouse does not support the long file names of the OS/2 High Performance File System (HPFS). CM Mouse will run on HPFS file systems, but it will be unable to read files that do not conform to the “8.3” naming convention.
- CM Mouse does minimal checking on numeric values in button definitions. In general, if a numeric value contains a non-numeric character, the error is flagged. It is assumed, however, that numeric values are within reasonable ranges (approximately -32000 to +32000).
- CM Mouse does not check that an AREA definition is valid (for example, the starting row/column is less than the ending row/column). If an AREA is defined which has a starting row or column greater than the ending row or column, a *null* AREA is defined, and the definition is never used since the mouse always fails the test of being within the AREA.

Appendix A. CM Mouse Keyword Reference

Table 3. CM Mouse Keyword Reference

Keyword	**Description**	**Runtime**	**Environments**
{beep}	Sound short beep	-	All
{clip cut} {clip cut textonly}	CUT marked data from host to clipboard	-	All
{clip copy} {clip copy textonly}	COPY marked data from host to clipboard	-	All
{clip copyappend}	Append marked data to clipboard	-	OS2
{clip from <r1> <c1> <r2> <c2>}	Copy specified area of host screen to clipboard	-	All
{clip paste}	Begin a PASTE clipboard operation	-	All
{clip to <r> <c>}	Copy clipboard text to specified area of host screen	-	All
{clip place}	Place a pending clipboard paste operation	-	OS2
{clip cancel}	Cancel a pending clipboard paste operation	-	OS2
{clip clear}	Clear data in the marking box	-	All
{clip undo}	Undo a clipboard paste operation	-	All
{clip unmark}	Remove marking box	-	WIN
{dde <function> <parameters>}	DDE Commands	-	OS/2
{editmmm}	Copy menu file and edit it	-	All
{hostwait n}	Wait *xxx* seconds for keyboard unlock	-	All
{if <condition> {then}script1{else}script2}	Conditional script execution	-	All
{lock on} {lock off}	Lock or unlock the keyboard	-	All
{map}	Display screen map for current session	-	OS2
{mmenu}	Show CM Mouse control panel	-	All
{mrowcol x y}	Move mouse cursor to row *x*, column *y*	-	All
{null}	Do nothing	-	All
{pause n}	Pause (wait) for *n* milliseconds	-	All
{pfkey}	Send PFkey/ENTER left of mouse cursor	-	All

Table 3. CM Mouse Keyword Reference (continued)

Keyword	**Description**	**Runtime**	**Environments**
{pfkey first}	Send first PFkey/ENTER on the line	-	All
{pfkey last}	Send last PFkey/ENTER on the line	-	All
{pfkeyrev} {pfkeyrev first} {pfkeyrev last}	Reverse format of {pfkey}	-	All
{pfkeyrev} {pfkeyrev first} {pfkeyrev last}	Reverse format of {pfkey first}	-	All
{pfkeyrev} {pfkeyrev first} {pfkeyrev last}	Reverse format of {pfkey last}	-	All
{printscreen LPTx}	Print the host screen a PC printer	-	All
{printscreen Fname} {printscreen Fname APPEND}	Copy or append the host screen to a PC file	-	All
{quit}	Terminate script execution	-	All
{rowcol x y}	Move host cursor to row *x*, column *y*	-	All
{seek}	Move host cursor to mouse cursor position	-	All
{unseek}	Move mouse cursor to host cursor position	-	All
{search FOR ’string’ AT r1 c1 r2 c2 WAIT n NOT ASIS NOQUIT}	Search host screen for a string	-	All
{seekelse}	Move host cursor, optionally abort	-	All
{seekcol x}	Move host cursor to mouse row, column *x*	-	All
{seekrow x}	Move host cursor to mouse column, row *x*	-	All
{set Name Value}	Set the value of a CM Mouse variable	-	All
{switchto <session>|*|next|prev }	Switch to another host session	-	All
{sys <cmd> <parms>}	Run a system command	-	All
{sysmenu}	Pulldown emulator system menu	-	All
{win <session>|*|prev|next MIN| MAX| RESTORE| HIDE| SHOW| ACTIVATE| DEACTIVATE}	Manipulate specified emulator window	-	All
{xfer ..}	Transfer file	-	All
{?<text>}	Pop-up *<text>* to confirm or abort	-	All
&{break}	Substitute a line break character (hex ’0d’)	-	All

Table 3. CM Mouse Keyword Reference (continued)

Keyword	Description	Runtime	Environments
&{chars r c l} or {&chars r c l}	Chars at row=*r*, col=*c* for length=*l*	✓	All
&{editor}	Name of editor (editor=)	-	All
&{env VarName}	Value of environment variable *VarName*	-	All
&{hcol} or {&hcol}	Current col of host cursor	✓	All
&{hrow} or {&hrow}	Current row of host cursor	✓	All
&{hour} &{min} &{sec}	Current time values (24-hour clock)	-	All
&{month} &{day} &{year}	Current date values	-	All
&{math 'val1' +|-|/|* 'val2'}	Perform mathematical operation	-	All
&{mmm}	Name of current pop-up menu file	-	All
&{mcol} or {&mcol}	Current col of mouse cursor	✓	All
&{mrow} or {&mrow}	Current row of mouse cursor	✓	All
&{num} or {&num}	Numeric string nearest to mouse cursor	✓	All
&{num first} or {&num first}	First numeric string on the line	✓	All
&{num last} or {&num last}	Last numeric string on the line	✓	All
&{num at <row> <col>} or {&num at <row> <col>}	Numeric string nearest row/column specified	✓	All
&{popup <menuname>} or {&popup <menuname>}	Display a pop-up menu, substitute selected line	✓	All
&{sid} or {&sid}	Current host session id (A,B,C...)	-	All
&{rows}	Rows in host screen	-	All
&{cols}	Columns in host screen	-	All
&{srow} &{scol} or {&srow} {&scol}	Row of last successful {search...}	✓	All
&{srow} &{scol} or {&srow} {&scol}	Column of last successful {search...}	✓	All
&{str <function> <parms>}	Perform string functions	✓	All
&{var Name} or {&var Name}	Value of a CM Mouse variable	✓	All
&{word delimit|include '<chars>'} or {&word}	Alphanumeric word nearest to mouse cursor	✓	All
&{word first} or {&word first}	First word on the line of the mouse cursor	✓	All
&{word last} or {&word last}	Last word on the line of the mouse cursor	✓	All
&{word at <row> <col>} or {&word at <row> <col>}	Word nearest to the row and column specified	✓	All

Table 3. CM Mouse Keyword Reference (continued)

Keyword	**Description**	**Runtime**	**Environments**
&{?<Qtext>|<Atext>} or {&?<Qtext>|<Atext>}	Prompt user for input using dialog	✓	All
&{rexx PgmSource} or {&rexx PgmSource}	Run a REXX program and substitute EXIT value	✓	OS2
[enter]	3270/5250 ENTER key	-	All
[pf1]-[pf24] [pf01]-[pf09]	3270/5250 Program Function keys	-	All
[newline] [tab] [backtab]	3270/5250 cursor movement keys	-	All
[home] [up] [down] [right] [left]	3270/5250 cursor movement keys	-	All
[fastright] [fastleft]	3270/5250 cursor movement keys	-	All
[pa1] [pa2] [pa3] [attn]	3270 Program Attention keys	-	All
[eraseeof] [delete] [insert]	Misc 3270 keys	-	All
[backspace] [clear] [crsel]	Misc 3270 keys	-	All
[reset] [sysreq]	Misc 3270 keys	-	All
[test]	3270 TEST key	-	All
[fieldmark] [altcsr]	Misc 3270 keys	-	All
[erinp]	Misc 3270 keys	-	All
[fldext] [hex]	Misc 5250 keys	-	OS2
[worddel]	Delete word at host cursor	-	All
[wordright] [wordleft]	Move host cursor 1 word right/left	-	All
[eof]	Move host cursor to end of field	-	All
[pageup] [pagedn]	5250 scroll keys	-	OS2
[help]	5250 help key	-	OS2

Appendix B. BDF File Syntax Diagrams

```
►►──SCREEN──Row──Column──┬─EXACT──┬──┬─A────┬──"──FindString──"──┬─────────┬──►◄
                         ├─LINE───┤  ├─B────┤                    └─Comment─┘
                         └─SCREEN─┘  ├─C────┤
                                     ├─...──┤
                                     ├─3270─┤
                                     ├─5250─┤
                                     └─*────┘
```

```
►►──AND-SCREEN──Row──Column──┬─EXACT──┬──┬─A────┬─────────────────────────────►
                             ├─LINE───┤  ├─B────┤
                             └─SCREEN─┘  ├─C────┤
                                         ├─...──┤
                                         ├─3270─┤
                                         ├─5250─┤
                                         └─*────┘

►─"──FindString──"──┬─────────┬───────────────────────────────────────────────►◄
                    └─Comment─┘
```

```
►►──OR-SCREEN──Row──Column──┬─EXACT──┬──┬─A────┬──────────────────────────────►
                            ├─LINE───┤  ├─B────┤
                            └─SCREEN─┘  ├─C────┤
                                        ├─...──┤
                                        ├─3270─┤
                                        ├─5250─┤
                                        └─*────┘

►─"──FindString──"──┬─────────┬───────────────────────────────────────────────►◄
                    └─Comment─┘
```

```
►►──AREA──StartRow──StartColumn──EndRow──EndColumn──┬─────────┬───────────────►◄
                                                    └─Comment─┘
```

```
►►──BUTTON──┬─LEFT──────────┬─"──ButtonDefinition──"─┬─────────┬──────────►◄
            ├─RIGHT─────────┤                        └─Comment─┘
            ├─MIDDLE────────┤
            ├─LEFT+RIGHT────┤
            ├─LEFT+MIDDLE───┤
            ├─RIGHT+LEFT────┤
            ├─RIGHT+MIDDLE──┤
            ├─MIDDLE+LEFT───┤
            ├─MIDDLE+RIGHT──┤
            ├─DBLLEFT───────┤
            ├─DBLRIGHT──────┤
            └─DBLMIDDLE─────┘
```

```
►►──DRAG──┬─FILE────┬─"──ButtonDefinition──"─┬─────────┬────────────────►◄
          ├─PRINT───┤                        └─Comment─┘
          └─DISCARD─┘
```

```
►►──DROP──FILE──"──ButtonDefinition──"─┬─────────┬──────────────────────►◄
                                       └─Comment─┘
```

```
►►──SET──VariableName─┬─────────┬─"──VariableValue──"─┬─────────┬───────►◄
                      └─LITERAL─┘                     └─Comment─┘
```

```
►►──INCLUDE─┬───────────────┬─FileName─┬─────────┬──────────────────────►◄
            └─RelativePath\─┘          └─Comment─┘
```

```
►►──DOIF──'string1' ─┬─=───┬─'string2' ─────────────────────────────────►◄
                     ├─==──┤
                     ├─!=──┤
                     ├─!==─┤
                     ├─<───┤
                     ├─>───┤
                     ├─<=──┤
                     ├─>=──┤
                     ├─=#──┤
                     └─!=#─┘
```

```
►►──ENDIF───────────────────────────────────────────────────────────────►◄
```

```
►►──DEFINE──┬─NAMEMAP─┬─────────────────────────────────────────────────►◄
            └─MapType─┘
```

```
►►──ENDDEFINE──────────────────────────────────────────────────────────────►◄


►►──MAP───┬─OFF──────────────┬──┬─────────┬────────────────────────────────►◄
          ├─CHORD TO DCLICK──┤  └─Comment─┘
          └─DCLICK TO CHORD──┘


►►──*───┬─────────┬────────────────────────────────────────────────────────►◄
        └─Comment─┘
```

Appendix C. MMM File Syntax Diagrams

```
►►──TITLE──"──Title──"──┬─────────┬──────────────────────────────────►◄
                        └─Comment─┘
```

```
►►──COLORS──┬──────────┬──┬──────────┬──┬──────────┬──┬──────────┬──►◄
            │  (1)     │  │  (2)     │  │  (3)     │  │  (4)     │
            ├──────────┤  ├──────────┤  ├──────────┤  ├──────────┤
            ├─BLACK────┤  ├─BLACK────┤  ├─BLACK────┤  ├─BLACK────┤
            ├─GRAY─────┤  ├─GRAY─────┤  ├─GRAY─────┤  ├─GRAY─────┤
            ├─BLUE─────┤  ├─BLUE─────┤  ├─BLUE─────┤  ├─BLUE─────┤
            ├─GREEN────┤  ├─GREEN────┤  ├─GREEN────┤  ├─GREEN────┤
            ├─CYAN─────┤  ├─CYAN─────┤  ├─CYAN─────┤  ├─CYAN─────┤
            ├─RED──────┤  ├─RED──────┤  ├─RED──────┤  ├─RED──────┤
            ├─MAGENTA──┤  ├─MAGENTA──┤  ├─MAGENTA──┤  ├─MAGENTA──┤
            ├─YELLOW───┤  ├─YELLOW───┤  ├─YELLOW───┤  ├─YELLOW───┤
            ├─WHITE────┤  ├─WHITE────┤  ├─WHITE────┤  ├─WHITE────┤
            ├─DGRAY────┤  ├─DGRAY────┤  ├─DGRAY────┤  ├─DGRAY────┤
            ├─DBLUE────┤  ├─DBLUE────┤  ├─DBLUE────┤  ├─DBLUE────┤
            ├─DGREEN───┤  ├─DGREEN───┤  ├─DGREEN───┤  ├─DGREEN───┤
            ├─DCYAN────┤  ├─DCYAN────┤  ├─DCYAN────┤  ├─DCYAN────┤
            ├─DRED─────┤  ├─DRED─────┤  ├─DRED─────┤  ├─DRED─────┤
            ├─DMAGENTA─┤  ├─DMAGENTA─┤  ├─DMAGENTA─┤  ├─DMAGENTA─┤
            └─DYELLOW──┘  └─DYELLOW──┘  └─DYELLOW──┘  └─DYELLOW──┘
```

Notes:

1. Foreground menu color.
2. Background menu color.
3. Foreground bounce bar color.
4. Background bounce bar color.

```
►►──PLACE──┬─Row─┬──┬─Column─┬───────────────────────────────────────►◄
           └─*───┘  └─*──────┘
```

```
►►──BAR──┬─InitialPosition─┬─────────────────────────────────────────►◄
         └─KEEP────────────┘
```

```
►►──LINE──┬───────────┬──┬───────────┬──────────────────────────────────►
          │   (1)     │  │   (2)     │
          ├───────────┤  ├───────────┤
          ├─BLACK─────┤  ├─BLACK─────┤
          ├─GRAY──────┤  ├─GRAY──────┤
          ├─BLUE──────┤  ├─BLUE──────┤
          ├─GREEN─────┤  ├─GREEN─────┤
          ├─CYAN──────┤  ├─CYAN──────┤
          ├─RED───────┤  ├─RED───────┤
          ├─MAGENTA───┤  ├─MAGENTA───┤
          ├─YELLOW────┤  ├─YELLOW────┤
          ├─WHITE─────┤  ├─WHITE─────┤
          ├─DGRAY─────┤  ├─DGRAY─────┤
          ├─DBLUE─────┤  ├─DBLUE─────┤
          ├─DGREEN────┤  ├─DGREEN────┤
          ├─DCYAN─────┤  ├─DCYAN─────┤
          ├─DRED──────┤  ├─DRED──────┤
          ├─DMAGENTA──┤  ├─DMAGENTA──┤
          └─DYELLOW───┘  └─DYELLOW───┘

►─"──┬────────────┬──"──┬──────────────────┬──"──────────────────────────►◄
     └─MenuText───┘     └─SubstitutionText─┘
```

Notes:

1. Foreground color.
2. Background color.

```
►►──SET──VariableName──┬─────────┬──"──VariableValue──"──┬─────────┬──────►◄
                       └─LITERAL─┘                       └─Comment─┘


►►──DOIF──'string1'──┬─=────┬──'string2'──────────────────────────────────►◄
                     ├─==───┤
                     ├─!=───┤
                     ├─!==──┤
                     ├─<────┤
                     ├─>────┤
                     ├─<=───┤
                     ├─>=───┤
                     ├─=#───┤
                     └─!=#──┘


►►──ENDIF─────────────────────────────────────────────────────────────────►◄


►►──*──┬─────────┬────────────────────────────────────────────────────────►◄
       └─Comment─┘
```

Appendix D. Notices

This information was developed for products and services offered in the United States. IBM may not offer the products, services, or features discussed in this information in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this information. The furnishing of this information does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
500 Columbus Avenue
Thornwood, NY 10594
U.S.A

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS INFORMATION AS IS WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or program(s) described in this information at any time without notice.

Any references in this information to non-IBM documentation or non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those documents or Web sites. The materials for those documents or Web sites are not part of the materials for this IBM product and use of those documents or Web sites is at your own risk.

Licensees of this program who wish to have information about it for the purpose of enabling:(i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Department T01
Building 062
P.O. Box 12195
Research Triangle Park, NC 27709-2195

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries, or both:

IBM
OS/2
Office Vision
AS/400
Global Network
Presentation Manager
GDDM

Microsoft, Windows, Windows NT©, and the Windows logo are registered trademarks of the Microsoft Corporation.

Glossary

A

AREA Definition Statement. A statement which appears in a CM Mouse BDF file and begins with the text AREA. Used by CM Mouse to subdivide a particular screen into rectangular areas.

B

BAR Statement. A menu file statement which defines the initial item on which the bounce bar cursor appears.

Button Definition File (BDF). A file which, when read, tells CM Mouse how to recognize host screens and what to do when a mouse button is pressed. For instance, what keystrokes are to be sent to the host, what pop-up menu is to be displayed, or what special functions are to be performed.

Button Definition. A generic term used to refer to a string of characters which contains host keystrokes and/or CM Mouse control and substitution keywords.

BUTTON Definition Statement. A statement which appears in a CM Mouse BDF file and begins with the text BUTTON. Used to identify what keystrokes or special functions are to be associated with a particular mouse button.

C

CM Mouse. A program which runs with an emulation program on a PC and allows you to use a workstation-attached mouse for host-based applications.

Clipboard. The “cut and paste” and “copy and paste” functions provided by the emulator or window system. These functions are available when you use CM Mouse.

COLORS Statement. A menu file statement which defines the colors to be used for a pop-up menu.

Communications Manager. A general term referring to the communications control program. Also the proper name of the IBM OS/2 communications product.

Control Panel. A special window which allows the user to change various CM Mouse operating parameters such as directory names and options.

Control Words. Special CM Mouse commands which perform actions specifically related to the mouse.

G

GDDM Graphics. Refers to a class of host applications which use the GDDM program product to present graphics and text.

Global Default Area. A series of AREA statements which appear in the BDF file before the first SCREEN statement. These AREA statements are used if no AREA within a SCREEN definition is matched.

Global Default Buttons. Button definitions which appear in the BDF file before the first SCREEN statement. These button definitions are used if no SCREEN definition in the BDF file matches the current host screen.

H

Host Control Words. Instructions to the emulator program to interact with the host or to control the emulator itself.

I

INCLUDE Statement. This statement can appear anywhere in a button definition file. It can be used to break up large button definition files into logical sections.

ISPF. A host application (Interactive System Productivity Facility).

Keyword. A delimited string which is recognized by CM Mouse to have special meaning. Keywords are typically delimited by either square or curly braces. A list of all CM Mouse keywords is shown in "Appendix A. CM Mouse Keyword Reference" on page 147.

L

LINE Statement. A menu file statement which defines a single selectable item in a pop-up menu list.

M

MMM File. A CM Mouse menu file used to define the appearance of a pop-up menu including the text of each selectable item on the menu, menu colors, and the title line.

Mouse Pointer. Small graphic shape which moves on the computer display when the mouse is moved.

O

OS/2 EE Communications Manager. A component of the OS/2 operating system which provides various communications functions, including host 3270 and 5250 terminal emulation.

P

Personal Communications/3270 (PC/3270). An IBM software product that emulates a 3270 terminal display. Versions are available for both Windows and native DOS (character) mode.

PLACE Statement. A menu file statement which determines where the pop-up menu is placed on the host screen.

Presubstitution Words. CM Mouse substitution words processed before the button definition is executed.

Primary Button Definition File. The file read by CM Mouse during initialization and whenever the action bar items are selected from the CM Mouse control panel. The primary BDF file usually contains several INCLUDE statements to cause other BDF files to be read as well.

PROFS/Office Vision. A host application with administrative organization features.

R

Runtime Substitution Words. Substitution words processed at the time the button definition is executed.

S

SCREEN Definition Statement. A statement which appears in a BDF file and begins with the text SCREEN. Used by CM Mouse to identify particular host screens by matching some part of the host screen with a string.

SET Statement. A statement which can appear anywhere in a button definition file or a menu file. This statement contains a variable value which can be used with only presubstitution words.

Substitution Words. Words used in several statements in BDF and MMM files. These substitution words are replaced with a value during execution so that you can construct definitions which depend on runtime values such as the position of the mouse or host cursors.

T

TITLE Statement. A menu file statement which defines an optional title for a pop-up menu.

V

VM RDRLIST. A VM host application function which receives incoming online mail.

W

Windows. A graphical user interface which runs on DOS. Also a generic term for an area of the screen that an application uses to display output and get user input.

X

x3270. Another name for the IBM 3270 Emulator for the X Window Syste.

Index

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.