Getting Started With Your DOMAIN/IX System

Order No. 008017
Revision 00
Software Release 9.2

Apollo Computer Inc.
330 Billerica Road
Chelmsford, MA 01824

First Printing: February, 1986

This document was produced using the Interleaf Workstation Publishing Software (WPS). Interleaf and WPS are trademarks of Interleaf, Inc.

Preface

Getting Started With Your DOMAIN/IX System introduces you to the basic concepts you'll need to use our implementation of the UNIX™ operating system. It teaches you how to use the keyboard, manage the information displayed on your screen, and manipulate text. You'll also learn how to request system services using interactive commands.

Whether you are new to the DOMAIN® system or not, you can soon get started performing important UNIX functions with ease. In fact, we've written the material so that anyone who understands UNIX "basics" can apply that knowledge to our flexible operating environment in a very short time. You'll benefit from the improved user interface and excellent time–saving UNIX system extensions that we supply with the DOMAIN/IX™ product.

Because this book is targeted for first–time users of DOMAIN/IX software, we carefully define terms and try to avoid computer industry jargon. We have also included examples you can try at your computer while you read. In this way, you can learn by doing. Once you finish reading the book and trying all the examples, you will have an easy–to–read overview of the DOMAIN/IX system.

The Organization of this Manual

We've organized the information in this manual as follows:

Chapter 1	Describes your keyboard and display, and explains how DOMAIN/IX programs are made available on your node.
Chapter 2	Explains log–in procedures, and helps you understand your display's appearance once you have logged on to the system.

Chapter 3	Explains the rules for entering commands that perform basic DOMAIN/IX functions.
Chapter 4	Provides a detailed description of how you can manage and control the elements of your display.
Chapter 5	Tells how the DOMAIN/IX system organizes information.
Chapter 6	Teaches you how to create, edit, and read text. Also explains how to print, copy, and delete information as necessary.
Chapter 7	Gives a brief overview of UNIX Shells. Also describes pipes and filters, redirection of input and output, and Shell scripts.
Chapter 8	Describes what you need to know in order to make the transition between the two versions of our UNIX implementation.
Keyboard Control Summary	Provides a short graphic overview of keyboard usage.
DM Environment Variable Settings	Describes important variables that affect the way that your display handles data.
Error Message Summary	Explains some common messages that typically appear as the result of a user error.
Command Synopsis	Summarizes commands and keys described in this manual.
Glossary	Defines terms used in this manual.

Documentation Conventions

Unless otherwise noted in the text, this manual uses the following symbolic conventions.

example	Color words in command examples represent literal user keyboard input.

command	Bold, lowercase words represent commands or keywords that you must use literally.
glossary term	Italicized terms are defined in the Glossary.
“filename”	Double quotation marks enclose the name of an object or term used in an example.
<RETURN>	Angle brackets enclose the name of a key on the keyboard.
CTRL/D	The notation CTRL/ followed by the name of a key indicates a control character sequence. Hold down <CTRL> while typing the character.

Related Manuals

When you complete this manual, continue with the *DOMAIN/IX User's Guide* (005803). It provides advanced information about performing various tasks using system components. In particular, it contains extensive material on UNIX Shells, software development tools, and the communications utilities. If you need to know more about DOMAIN/IX text processing and editing capabilities, consult the *DOMAIN/IX Text Processing Guide* (005802).

The *DOMAIN System User's Guide* (005488) describes the special functions of the Display Manager and other enhanced capabilities offered by the DOMAIN system.

The *DOMAIN/IX Command Reference for BSD4.2* (005800) and the *DOMAIN/IX Command Reference for System V* (005798) describe all user–oriented Shell commands supported, respectively, by the BSD4.2 and System V versions of the UNIX operating system. The commands are arranged alphabetically for quick and easy access.

The *DOMAIN/IX Programmer's Reference for BSD4.2* (005801) and the *DOMAIN/IX Programmer's Reference for System V* (005799) describe the system calls and library functions supported, respectively, by the BSD4.2 and System V versions of the UNIX operating system. The system calls are also arranged alphabetically.

Problems, Questions, and Suggestions

We appreciate comments from the people who use our system. In order to make it easy for you to communicate with us, we provide the User Change Request (UCR) system for software–related comments, and the Reader's Response form for documentation comments. By using these formal channels you make it easy for us to respond to your comments.

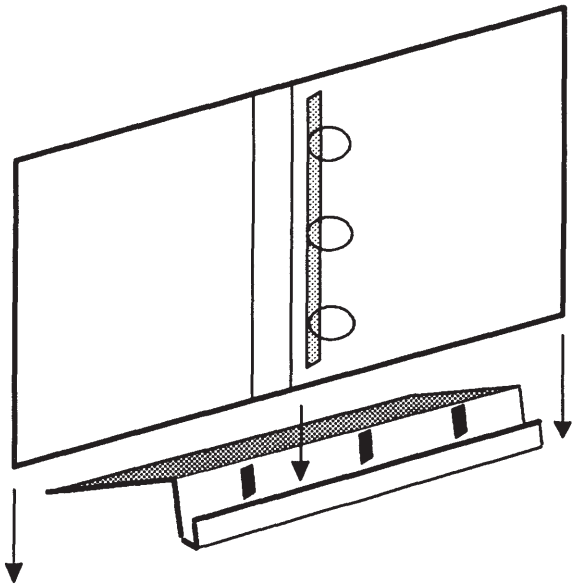
You can get more information about how to submit a UCR by consulting the DOMAIN System Command Reference (002547). You can view the same description on–line by typing the following at a UNIX Shell prompt:

```
% /com/help crucr <RETURN>
```

For your documentation comments, we've included a Reader's Response form at the back of each manual.

Using the Stand–Up Binder

The plastic page lifter is designed to function as an easel for propping up the binder on your desktop. The following illustration shows how to use it.



Contents

Chapter 4 Managing Windows and Pads

Chapter 5 Organizing Information

Chapter 6 Using Files

Illustrations

Tables

Chapter 1

Introduction

The DOMAIN/IX product is an implementation of the UNIX operating system that runs on *DOMAIN* computers (or *nodes*) linked together with a local area network. Figure 1–1 shows the wide variety of DOMAIN nodes that may be connected in this network.

Each node can use the data and programs of other network nodes. Each contains main memory, and may have its own disk, or share one with another node (i.e., be considered a *diskless node*).

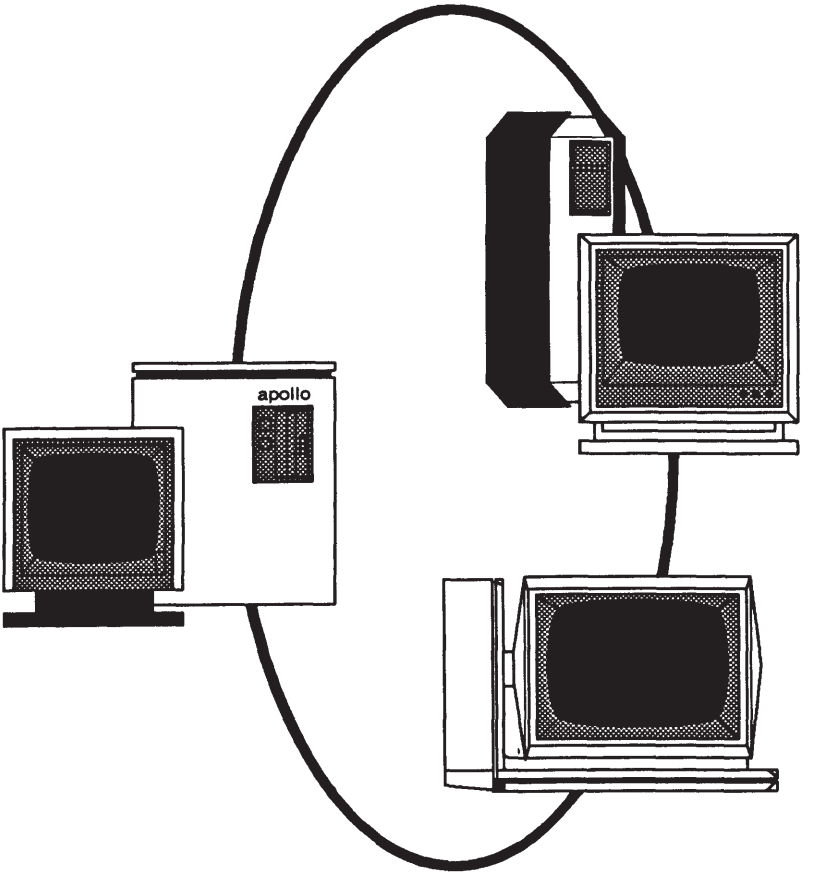


Figure 1–1. DOMAIN System Nodes

DOMAIN/IX software supports the DOMAIN distributed file system, ring network, and bit–mapped, high–resolution displays. In addition to bringing the benefits of a networked architecture to the UNIX system, DOMAIN/IX software offers many features that are seldom

found on either time–sharing or workstation implementations of this software.

We provide two versions of the DOMAIN/IX system to our customers. The *sys5* version supports UNIX System V, Release 2 from AT&T Bell Laboratories. The *bsd4.2* version supports 4.2 BSD from the University of California at Berkeley.

This chapter introduces the DOMAIN/IX product and explains the basic components of your DOMAIN node. The best way to master the material in this book is to try each example at your node while you read. In the back of this book, there is a glossary that defines terms specific to the DOMAIN/IX system. The glossary also lists terms used throughout our documentation.

Getting to Know Your DOMAIN Node

The node you're using includes a keyboard and a color or black–and–white display screen. Display management software lets you create up to 16 different *windows* on the screen. Each window is a separate computing environment in which you can execute programs, edit text, or read text. You can move the windows anywhere on your screen, change their size and shape, and overlap or shuffle them as you might papers on your desk.

Your Keyboard

Most likely, your node is equipped with a *low–profile keyboard*. If your keyboard is not like the one shown in Figure 1–2, you could have an older model, the *880 keyboard*. Because the information we present in our exercises assumes that you have a low–profile keyboard, you must refer to Appendix A for equivalent 880 keyboard definitions and control sequences.

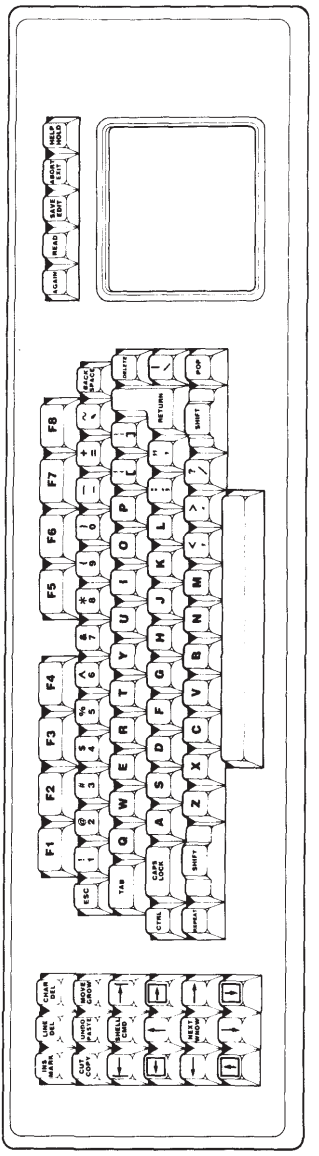


Figure 1–2. A Low–Profile Keyboard

Your Display

The display you're using probably resembles the *landscape* (horizontal) display in Figure 1–3. You may, however, have one of our older *portrait* (vertical) displays. If your screen looks blank, press any key to turn on the video display. The system automatically shuts off the video display if it is idle for more than 15 minutes.

If your screen still looks blank after you press a key, your node is not running. If you need help with starting your node, see your system administrator (the person responsible for system maintenance and security at your installation), or refer to the *Operator's Manual* for your particular node.

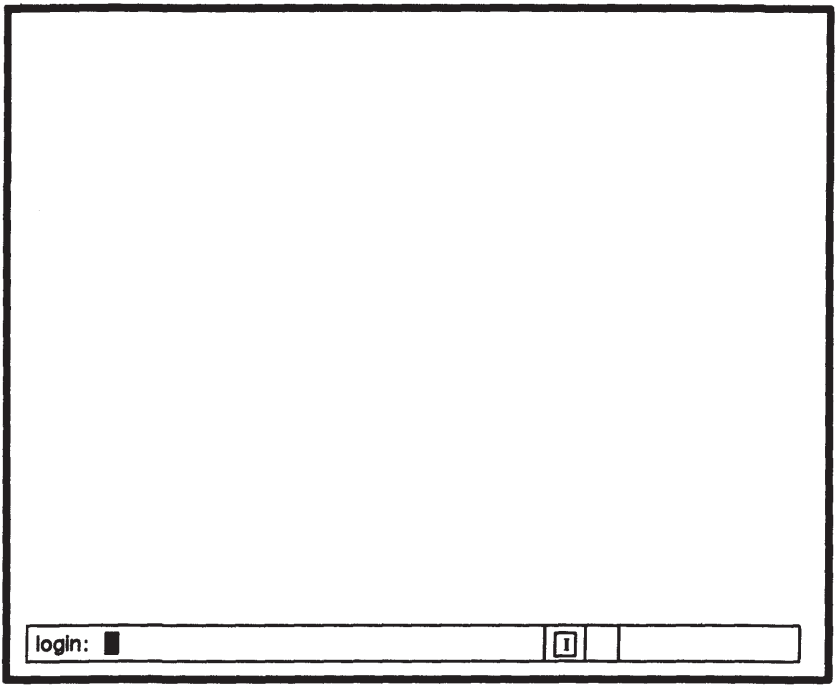


Figure 1–3. A Landscape Display

Notice the small, blinking box in the lower left corner of your display. This is the *cursor*. The box cursor indicates where the system will display the information that you type at your keyboard.

As you use the system, you may see two other cursor shapes: an arrow cursor (↑) and a cross in a circle cursor (⊕). The arrow cursor appears when you use the touchpad or mouse to move the cursor (see "Using the Touchpad" and "Using the Mouse" later in this chapter). The circle cursor appears when the Display Manager is so busy that it cannot immediately respond to your commands. As a new user, you should simply wait until the circle cursor changes back to a blinking box or an arrow before you enter additional commands.

Moving the Cursor

Moving the cursor is the first step in learning to use the system. As you work through the examples in this book, you'll see that you must position the cursor at a specific location on your display before giving an instruction to the system.

To move the cursor, use the arrow keys, or a mouse. The touchpad and the mouse are optional pieces of equipment. The arrow keys are available on all keyboards.

As you read about the ways to move the cursor in the next three sections, try using the keys, and the touchpad or mouse if you have one.

Using the Keyboard

You can use any of the keys highlighted in Figure 1–4 to move the cursor. To move the cursor using the *arrow keys* (← → ↑ ↓), press the arrow key that points in the direction you wish to move, and hold it down until the cursor reaches the desired destination.

To move to the beginning or end of a line displayed beneath the cursor, use the keys labeled ⇤ and ⇥ .

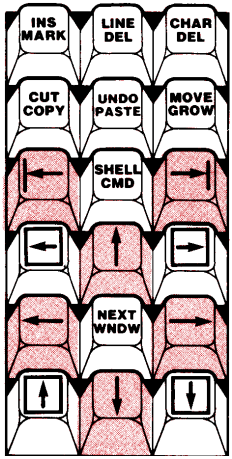


Figure 1–4. Keys that Move the Cursor

Using the Touchpad

The optional *touchpad* is located on the right side of your keyboard. (See Figure 1–5.) The touchpad is made of a pressure–sensitive material. When you press the material with your finger, the touchpad transmits data to the system, and the system moves the cursor.

While you're touching the material, the cursor appears as an arrow; when you lift your finger, the cursor reverts to a blinking box.

Take care not to puncture or scratch the touchpad's conductive material. Never use a sharp object (such as a pencil or pen point) on the touchpad. Scratches or punctures can move the cursor into undesired screen positions.

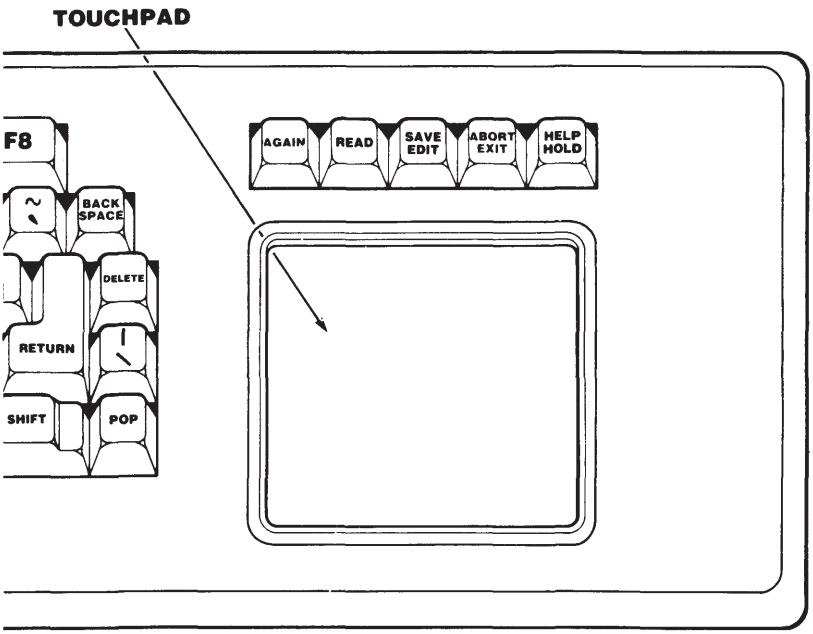


Figure 1–5. A Touchpad

Using the Mouse

The *mouse* is a small device that you move across a flat surface, such as your desktop or a pad of paper. (See Figure 1–6.) A ball in the base of the mouse detects motion, the mouse transmits data about the motion to the system, and the system moves the cursor.

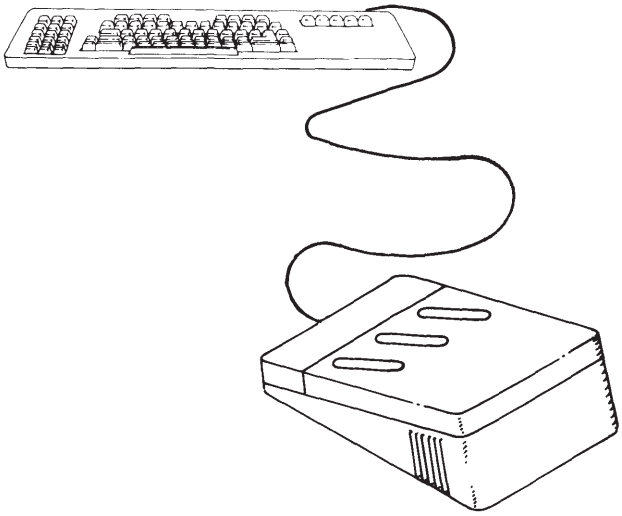


Figure 1–6. A Mouse

When you're moving the mouse, the cursor appears as an arrow; when you stop moving the mouse, the cursor reverts to a blinking box.

There are three keys on the mouse. You can use these keys to manipulate windows and read files. Chapter 4 explains how to expand, shrink, and shuffle windows with the mouse. Chapter 6 describes how you can use the mouse to examine the contents of files.

NOTE: The mouse keys are predefined to perform particular functions. You can, however, redefine the mouse keys to do other useful tasks. See the *DOMAIN System User's Guide* for details.

How Does It All Work?

DOMAIN/IX software combines the portability of the UNIX operating system with the flexible nature of the DOMAIN system's AEGIS operating system. Thus, these two entities co–reside in a partnership to supervise program execution on your node. As a result, the UNIX programs supplied as a part of the DOMAIN/IX system have the same file format as AEGIS programs. Furthermore, there is normally no distinction between processes that run UNIX programs on a DOMAIN node and those that run AEGIS programs.

As we describe DOMAIN/IX system operations, it is important for you to understand the two major components that are responsible for providing and monitoring program execution: the *Shell* and the *Display Manager* (*DM*).

The Shell Program

The Shell program provides access to traditional computing operations, such as printing documents, compiling and running programs, and monitoring system activities. The Shell "listens" for commands that you type at your keyboard. Shell commands invoke *utilities*, which are programs that perform the tasks you request.

As you read the material we present, you won't really need to know how the AEGIS Shell works. However, you should note that, by default, the system opens an AEGIS Shell when you first log in. You may, of course, arrange for a UNIX Shell to be opened instead. Or, you may decide to have both an AEGIS Shell and a UNIX Shell opened at log–in time. We will tell you more about this later, as well as how to issue commands from both types of Shells.

For now, remember that you can recognize the AEGIS Shell by the single dollar sign ($) it uses as its *prompt*. Programs use prompts to indicate that they are ready for you to type a command at the keyboard. The default prompt for all C Shells is a percent sign (%). The default prompt for a System V Bourne Shell is a pound sign (#); a BSD4.2 Bourne Shell uses a capital "B" followed by a dollar sign (B$).

Throughout this book, we use a C Shell for all our DOMAIN/IX system examples. Furthermore, we use the BSD4.2 version of the

DOMAIN/IX software. Our examples, however, do not depend on version or UNIX Shell type. Thus, whether you use the BSD4.2 or the System V version, whether you work in a C Shell or a Bourne Shell, you can still apply the necessary information and work through the examples.

The Display Manager

The Display Manager is a program that opens, closes, and moves windows on your screen. Furthermore, it can be used for changing other aspects of the display, such as background color and character font, and for creating and editing files, which we will explain more about later. The Display Manager also supervises the creation of computing environments (*processes*) in which you can execute programs. After you log in, the Display Manager creates a process in which a Shell program is running.

Summary

After reading this chapter, you should be familiar with your keyboard (and mouse or touchpad) and the basic information that your display provides. You should also know the fundamental purposes of Shell programs and the Display Manager. In the next chapter, you'll learn how to log in on your node so that you can begin using the DOMAIN/IX system.

Chapter 2

Logging In

Each time you log in, the DOMAIN/IX system executes programs that define your node's operating environment. The system uses various *start–up scripts* to initiate processes that must be running during the time that you are logged on to your node. One such script contains certain DM *environment variables* (described in Appendix B) that must be set in order for your node to be able to work according to the methods we demonstrate. If your display's behavior varies from our description, contact your system administrator. We do not recommend that you attempt to set your own environment at first.

Once you learn more about how the system works, you can tailor the operating environment on your node by modifying the scripts that the system uses at log–in time. (For example, you can arrange to have your own specific key definitions, default window positions, and tabs defined each time you log in.) We suggest that you, as a new user,

ask someone with previous experience to make any modifications other than those shown in this chapter.

Entering Your User ID and Password

When your node is running and the video display is turned on, a log–in prompt appears at the bottom of the screen. The prompt should read "login: ". (This is the standard UNIX system log–in prompt.)

NOTE: If your display shows the prompt as "Please log in: ", ask your system administrator to modify your DM environment so that the proper message appears.

The "login: " prompt indicates that the system is waiting for you to enter your *user ID* and *password*. If you don't know what names to enter, ask your system administrator (who defines a user account for every person authorized to use the system).

Your user account contains the name that the computer uses to identify you (user ID), as well as your password. If security is particularly important at your installation, user accounts may also contain project and organization names. The system uses this information to determine who can use the system and what resources they can use.

As we said earlier, the cursor indicates where the system will display the commands you type at your keyboard. Before logging in, move the cursor to the right of the log–in prompt.

To log in, first enter your valid user ID at the "login: " prompt. For example, if your user ID is "mary", the line would look like this:

```
login:  mary
```

If your system administrator says that project and organization names are required, you must also type these names. For example, if your project identifier is "newprog" and your organization is "systest", the line would look this way:

```
login:  mary.newprog.systest
```

If you make a mistake, press <BACKSPACE>. This key deletes characters as it moves the cursor back toward the beginning of the line.

When you have typed the required name(s) correctly, press <RETURN> to submit this information to the system. (Press <RETURN> whenever you wish to submit the line you've typed to the system.) The system accepts the line you submitted and requests your password.

After you have typed your user ID and hit <RETURN>, you will be prompted for your password. The system does not display the password you enter, but displays a dot for each character in the password. (Passwords may be a total of eight characters long, and may include both letters and numbers.) For example, if your password is "route 66", the following would be displayed:

```
Password:  ..... <RETURN>
```

If the system finds a user account that matches the names you supply, it logs you in and prints a "message of the day" at the bottom of your screen. By default, such a message might be:

```
**Welcome to DOMAIN/IX!**
```

If the system cannot find a user account that matches the names you supply, it first displays the following message at the bottom of your screen:

```
Login Incorrect
```

It then repeats the log–in prompt and displays the expected format of log–in information (l user ID [project [org]]). If you cannot get logged in, ask your system administrator for help.

Understanding Your Display After Log–In

Immediately after you log in, the Display Manager creates some windows on your screen. The position of these windows depends upon the type of display you're using. Figure 2–1 shows window positions on a landscape display.

In addition to presenting these windows, the Display Manager creates process(es) in which one or more specified Shells are running. The Shells created depend on how your DM environment is set. For example, we have arranged for our Display Manager to create both an AEGIS Shell and a C Shell. This is how our display will appear after we are logged in and all the necessary Shell processes have been created.

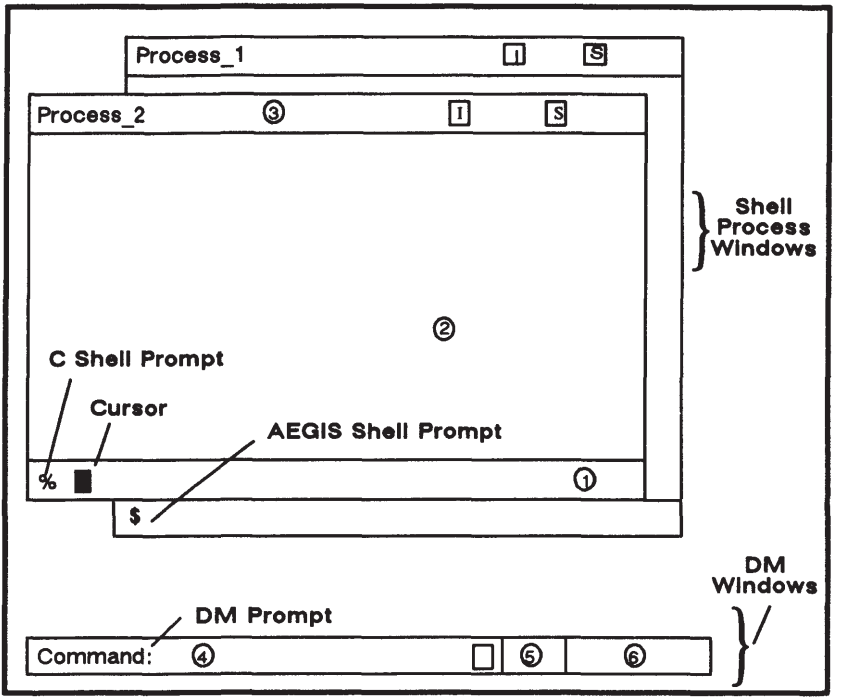


Figure 2–1. Window Positions on a Landscape Display

Notice that the cursor is in the lower left corner of your screen. When you logged in, you saw that the cursor position indicates where the system displays the commands you type. The cursor position also indicates which program (Shell or DM) receives your commands.

Before entering a command, always move the cursor into the appropriate window, depending on whether you are specifying a Shell or a DM command. To move from window to window, press <NEXT WNDW>. Depending on how many windows you have on your

display, you may have to press <NEXT WNDW> more than once to move the cursor to the window where you will execute the command.

When you pressed <NEXT WNDW>, you invoked a Display Manager command to do the task. Instead of typing the DM command that moves the cursor from window to window, you used a single key called a *DM function key*. We'll tell you more about using these special keys in the next chapter, when we discuss methods for invoking DM commands.

When you are ready to enter commands, remember to enter UNIX Shell and AEGIS Shell commands at their given prompts (shown in Figure 2–1), and DM commands at the "Command: " prompt. To move the cursor next to the Command: prompt, press <CMD>, another DM function key that you'll learn more about later.

Now let's look more closely at the parts of the initial display. Refer to Figure 2–1 as you read about the Shell process windows and the DM windows in the next few paragraphs. The numbers in the following list correspond to the numbers in the figure.

We'll begin with the Shell process windows.

1. The *process input window* for a C Shell contains the % prompt. After you type UNIX commands in this window, the C Shell reads the commands and invokes the appropriate utility.

   When you use this Shell to execute another program, that program may display its own prompt in the process input window. The prompt appears when the Shell is waiting for your next instruction.

2. The *process output window* is the large window above the process input window. It displays your commands (after you press <RETURN>) and the Shell's response to your commands. The response can be information you requested, a process status report, or an error message. The process output window is a transcript, or record, of your interaction with the system.

3. The *window legend* is the top border of the output window. It contains the process identification name (Process_xx) and the letters I (Insert) and S (Save). The letter "I" indicates that you can insert text in the input window rather than overstrike the existing command line. The letter "S"

specifies that you may scroll output one line at a time. Two other mode indicators are also used. The letter "H" indicates that the contents of the window are to be held when output is sent to the associated pad. The letter "A" tells you that the window will automatically enter hold mode. Chapter 4 of the *DOMAIN System User's Guide* explains these window mode indicators in greater detail.

The DM windows are at the bottom of your screen.

4. The *DM input window* contains the "Command: " prompt. In this window, you type DM commands. When you press <RETURN>, the DM reads these commands. Note the mode indicator "I" to the right of the DM input window. It tells you that the DM input window is operating in insert mode.

5. The *DM alarm window* displays a small pair of bells when a process displays a message in an output window that is hidden by an overlapping window. The "Responding to Alarms" section in Chapter 4 explains more about this window.

6. The *DM output window*, like the Shell output window, displays messages. However, the DM does not display the commands you enter.

Ending the Session — Logging Off

When you're ready to end the session, log off the system. Logging off prevents others from using your user account. Log off if your node is in a public place.

To log off, press <CMD> and type the **lo** (logout) command at the DM prompt as follows:

```
Command:  lo <RETURN>
```

You needn't worry about stopping active processes before logging off; the DM will do this for you. After logging you off, the DM redisplays the "login: " prompt.

Summary

After reading this chapter, you should be able to log in, interpret information on your display, and log out. In the next chapter, we'll explain how to enter Shell and DM commands.

Chapter 3

Entering Commands

If you were following the last chapter, you are no longer logged in to your node. Before you begin reading this chapter, you should log in once more. Now you are ready to try entering some commands. As you know, the examples in this book show a BSD4.2 C Shell prompt (%), as well as the DM prompt (Command:), where necessary. Recognizing these prompts should help you determine where to enter particular commands. Don't type the prompt itself; simply enter the command line shown in color.

You must type UNIX commands exactly as shown, including any capitalization. The DM and the AEGIS Shell do not differentiate between uppercase and lowercase characters, so you can enter AEGIS commands in any combination of cases. Remember to press <RETURN> at the end of each command line.

Invoking the Display Manager

There are three ways to enter Display Manager (DM) commands:

- Type the command in the DM input window (next to the "Command:" prompt).
- Press a specially–defined (and labeled) key, called a *DM function key*.
- Press a *control key sequence* (<CTRL> combined with another key).

Try typing a command in the DM input window. First, press <CMD> to move the cursor next to the "Command:" prompt. Now type

```
Command:  rs <RETURN>
```

After you press <RETURN>, the **rs** (refresh screen) command refreshes the entire screen. You'll see the display blink as the DM clears the screen and redraws windows.

DM Function Keys

You usually won't type DM command names at your keyboard. Instead, you will use certain single keys (DM function keys) that invoke DM commands. (We mentioned two of these keys, <NEXT WNDW> and <CMD> in the last chapter.)

Each function key highlighted in Figure 3–1 provides access to two commands. For example, the function key labeled CUT and COPY lets you delete (cut) text and copy text. To enter the command in the uppermost position (e.g., CUT), hold down <SHIFT> while you press the function key. To enter the command in the lowermost position (COPY, for example), simply press the function key.

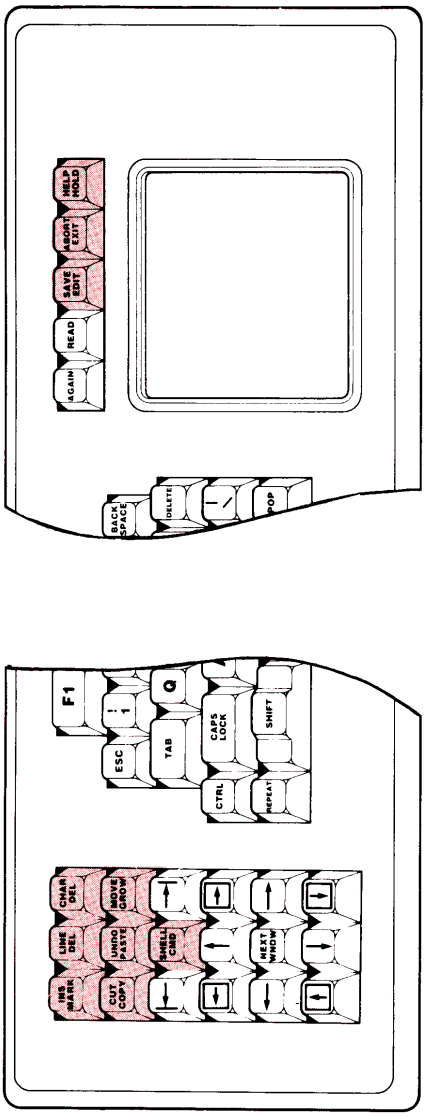


Figure 3–1. Function Keys that Access Two Commands

Control Key Sequences

Often you'll use *control key sequences* to invoke DM commands. To enter a control key sequence, hold down <CTRL> while you press another key.

Although you may want to use the DM function key <EXIT> to remove the window from your screen, you can also use the CTRL/N control key sequence to produce the same results. Try using CTRL/N to remove a Shell input window.

Creating a Process

An important function of the Display Manager is the creation and cancellation of Shell processes that you require during the time you are logged in.

You can create a new Shell process any time you need another computing environment. For example, if your initial Shell process is busy compiling a program, you can create a new Shell process to perform another task, such as formatting a text file.

Use <SHELL> to create a new process running a Shell program. The DM creates a new Shell process, along with all the necessary pads and windows. On your screen, you'll see input and transcript (output) pads as well as windows into those pads. The DM places the cursor next to the Shell prompt in the new process input window, and the process is ready to receive Shell commands.

NOTE: Until keys are redefined for UNIX functionality, the DM creates an AEGIS Shell when <SHELL> is pressed. See the next section for information on how to convert this process to create a Bourne or C Shell instead.

Redefining Keys For UNIX Functionality

With all installations of standard software, we supply the command files that define your keyboard. The file containing standard keyboard definitions for the low–profile keyboard is called "std_keys2".

With every DOMAIN/IX installation, we provide alternate versions of these standard key definitions – modified to provide UNIX system

functionality. These alternate key definitions reside in the files shown in Table 2–1. (The equivalent files for the 880 keyboard are named “sys5_keys” and “bsd4.2_keys”.)

Table 3–1. UNIX Key Definitions

Filename	Contents
sys5_keys2	System V UNIX keyboard definitions for the low–profile keyboard
bsd4.2_keys2	BSD 4.2 UNIX keyboard definitions for the low–profile keyboard

The BSD4.2 and the System V definitions files include commands that bind various keys to certain version–specific (or Shell–specific) features. Initially, none of these key–definition files are automatically invoked, although you may arrange for them to be.

To put any key–definition file into effect, execute the **cmdf** (command file) command at the Display Manager prompt, where the filename argument is one of the files mentioned earlier. For example,

Command: cmdf /sys/dm/bsd4.2_keys2 <RETURN>

invokes the BSD4.2 version UNIX key definitions on a low–profile keyboard. The following explains how the keys are redefined when the keyboard is remapped to the “bsd4.2_keys2” file:

<SHELL> This DM function key executes the DM command **cp /bin/start_csh**, which creates a C Shell and runs a personal UNIX log–in file (e.g., “.login”).

<TAB> When shifted, this key inserts a literal ASCII tab character.

<READ> This DM function key, which calls the DM to read a file, takes on a different prompt. The prompt becomes “read file: ” rather than “Read file: ”. Also, pathnames supplied as arguments become case–sensitive where they would not be otherwise (i.e., if “std_keys2” key definitions were in effect).

<EDIT> This DM function key, which calls the DM editor, takes on a different prompt. The prompt becomes “edit file: ” rather than “Edit file: ”. Also, pathnames become case–sensitive, where they would not be otherwise (i.e., if “std_ keys2” key definitions were in effect).

<CTRL/I> This control–key sequence generates an interrupt signal.

<CTRL/D> This control–key sequence produces an end–of–file (EOF) signal.

<CTRL/Z> This control–key sequence generates a suspend signal.

<CTRL/J> This control–key sequence breaks a previous suspend signal (produced by using CTRL/Z).

Invoking the System V version of UNIX key definitions on a low–profile keyboard (“sys5_keys2”) produces similar results. The <TAB>, <READ>, and <EDIT> keys, and the <CTRL/I> and <CTRL/D> control–key sequences, work in exactly the same manner as described above. The following describes the features that differ:

<SHELL> This key executes **cp /bin/start_sh**, which creates a Bourne Shell and runs a personal UNIX log–in file (e.g., “.profile”).

<CTRL/Z> Not applicable.

<CTRL/J> Not applicable.

If the keys on your keyboard don’t work as we describe, your system is probably reading a different set of key definitions. Ask your system administrator for help.

NOTE: Remaining examples in this book assume a keyboard redefined to "bsd4.2_keys2".

Using UNIX Commands

Now that your keyboard has been redefined, creating a UNIX Shell should be very easy. Hold down <SHIFT> while you press <SHELL>. This key issues the DM command that creates a new process running a Shell program. Since you've redefined your keyboard to "bsd4.2_keys2", the <SHELL> key creates a C Shell and displays the windows associated with the new Shell process.

NOTE: You can create a new Shell process at any time. You may use it to execute programs while your initial Shell process is busy performing other tasks.

Move the cursor next to the UNIX Shell prompt in the process input window and enter the **date** command. Remember to enter the command exactly as shown (i.e., in lowercase letters).

```
%  date <RETURN>
```

In the process output window, the Shell displays the command you typed and then invokes the utility program that displays the date and time. For example, the output of the command that you just typed might look like this:

```
Thurs  Feb 6 16:54:24  EDT 1986
```

Using AEGIS Commands

While AEGIS commands don't generally play a major part in this book, some important ones are used in our examples. They help demonstrate enhanced features of the DOMAIN/IX system that are not supplied by standard UNIX utilities. Each time that you execute a command beginning with **/com** from a UNIX Shell, you are invoking an AEGIS command.

NOTE: Without the **/com** prefix, you may execute these commands from an AEGIS Shell. If you do so, you should be aware that special characters (*wildcards*) sometimes used as shorthand in a UNIX Shell produce different results in an

AEGIS Shell. It is best not to use such devices with these commands until you understand the full effects of doing so.

Correcting Typing Errors

Don't worry if you make a typing mistake when you enter a command. Usually, you'll simply cause the system to display an error message on the screen. If you notice your mistake before you press <RETURN>, you can correct it by using one of the keys listed here:

- BACKSPACE
- CHAR DEL
- LINE DEL
- INS

As you learned earlier, <BACKSPACE> deletes characters as it moves the cursor back toward the beginning of the line. For example, if you type "date" and then press <BACKSPACE>, the letter "e" disappears.

<CHAR DEL> deletes the character at the current cursor position. For example, if you position the cursor on the "d" in "date" and press <CHAR DEL>, the "d" disappears.

<LINE DEL> deletes the entire line, no matter where the cursor is positioned on the line.

Notice the letter "I" in the Shell's window legend. The letter also appears to the right of the DM input window. It indicates that the DM and Shell input windows are operating in insert mode. As you may recall from Chapter 2, insert mode lets you change command lines in the input window by repositioning the cursor and inserting characters. The rest of the line moves right as you insert additional characters. The system inserts the text you enter rather than overstriking the existing command line.

To overstrike the command line, turn off insert mode by pressing <INS>. If the cursor is in the Shell input window, the letter "I" in the window legend disappears. If the cursor is in the DM input window, the letter "I" following the DM input window disappears. The

system then replaces existing characters with the new characters you type. You can turn insert mode back on by pressing <INS> again.

Getting Help

You can get information about available UNIX commands, system calls, and functions with the **man** command. This command lets you select and display on–line versions of reference material from the *DOMAIN/IX Command Reference* and the *DOMAIN/IX Programmer's Reference* for either version of DOMAIN/IX software. For example, to display the manual page for the **ls** command, type the following:

```
%  man ls <RETURN>
```

The **man** command opens a separate read window containing a formatted version of the manual page(s) for the **ls** command. While the manual page is displayed, you may continue to execute other UNIX Shell commands (including more **man** commands). When you are finished reading the manual page, use <EXIT> to close the window.

If you need help on DM or AEGIS commands, but cannot remember the name of the command that you want to use, you can view a complete listing of commands by pressing <HELP> (remember to hold <SHIFT> down at the same time). The prompt that appears in your DM input window should look like this:

```
Help on:
```

At this prompt, type the argument "commands" so that the line now appears as follows:

```
Help on:  commands  <RETURN>
```

If you know the name of the DM or AEGIS command, simply type that name at the prompt. For example, to get help on the **ap** (alarm pop) DM command, use this:

```
Help on:  ap <RETURN>
```

Summary

After reading this chapter, you should be able to create Shell processes, execute some simple commands, and get help on any of the commands that you use. In the next chapter, you'll learn how to better manipulate windows and pads.

Chapter 4

Managing Windows and Pads

Although you should be familiar with the concepts of windows and processes by now, this chapter describes how the Display Manager (DM) controls these entities on your display. Using DM function keys and control key sequences, you will learn how to:

- View hidden information in a window
- Shuffle windows
- Change the size of windows
- Create, suspend, and stop Shell processes

Looking Inside a Window

Windows let you view information stored in the system. *Pads* contain the information that you display. The information can be commands that you type, responses from the system, text files, or pictures.

In other words, a window provides a view of a pad. The window can present the entire pad, or only show part of the pad. Figure 4–1 illustrates a window over a pad, where only part of the pad is visible through the window.

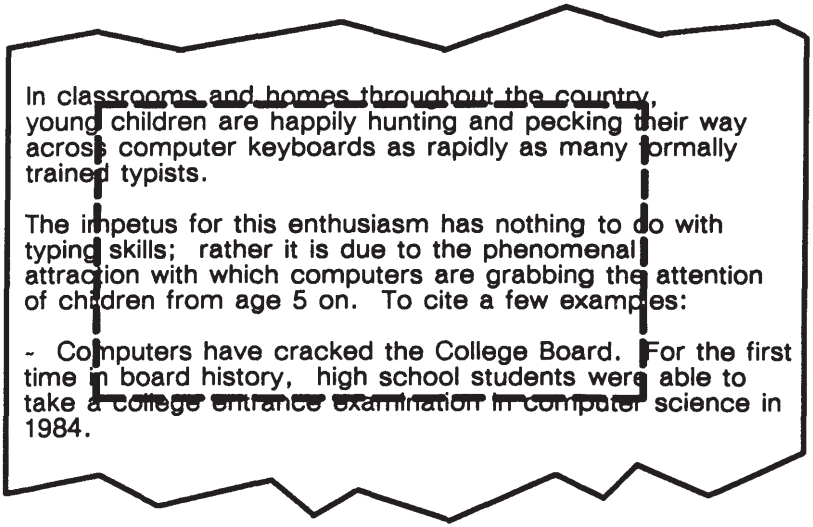


Figure 4–1. A Window Over a Pad

The dotted line in Figure 4–1 represents the window's edges. On the display, you can see only the characters inside the edges of the window. The rest of the pad is hidden. There are two ways to view the hidden parts of a pad: you can either move the pad under the window or enlarge the window. Later in this chapter, you'll learn both ways to view hidden information.

Windows have such attributes as size and position on the screen, position over a pad, and possibly color (if your node is a color model). You can imagine them as pieces of paper that you can stack and shuffle around on the screen just as you would if they were on

your desk. Later in this chapter, you'll learn how to manipulate windows, but first we'll describe the kinds of pads the DM uses and the keys that control them.

The DM provides three kinds of pads: input, transcript, and edit pads. *Input pads* accept the commands you type at your keyboard. In the last chapter, you typed commands through windows onto input pads.

Transcript pads (process output pads) keep a running record, or "transcript," of your interaction with programs by displaying your input and the program's output. You view transcript pads through process output windows. In our previous examples, the Shell displayed the commands you entered and its response to your commands on the transcript pad.

Figure 4–2 shows a process input pad for a C Shell, a DM input pad, and a transcript pad.

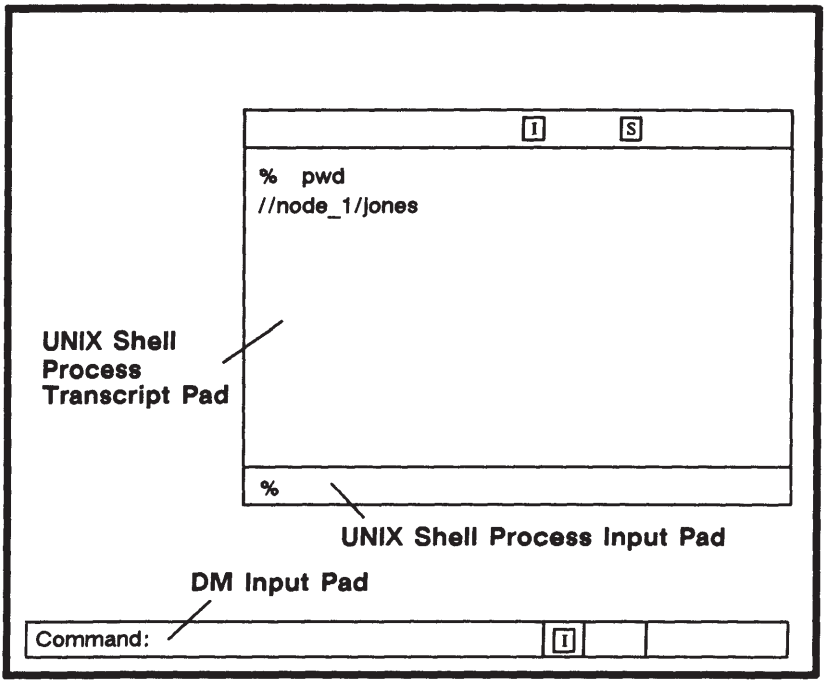


Figure 4–2. input and Transcript Pads

Edit pads contain copies of files that the DM displays when you press <READ> or <EDIT>. You can change text displayed on pads created

with <EDIT>. When you close the pad, the DM writes your changes to the permanent file. Pads created with <READ> are read–only; you can copy text from them, but you cannot add or delete text. Chapter 6 explains file reading and editing in detail.

To see what an edit pad and window look like on your display, you can read a text file supplied with your system. Press <READ>. The cursor moves into the DM input window and the DM displays the prompt "read file:".

To read a file called "/doc/domain_ix_release_notes.sr9.2", which describes the details of the SR9.2 DOMAIN/IX software version, type the filename and press <RETURN> as follows:

```
read file:  /doc/domain_ix_release_notes.sr9.2 <RETURN>
```

The DM creates a read–only window and pad to display the file "/doc/domain_ix_release_notes.sr9.2". (See Figure 4–3.)

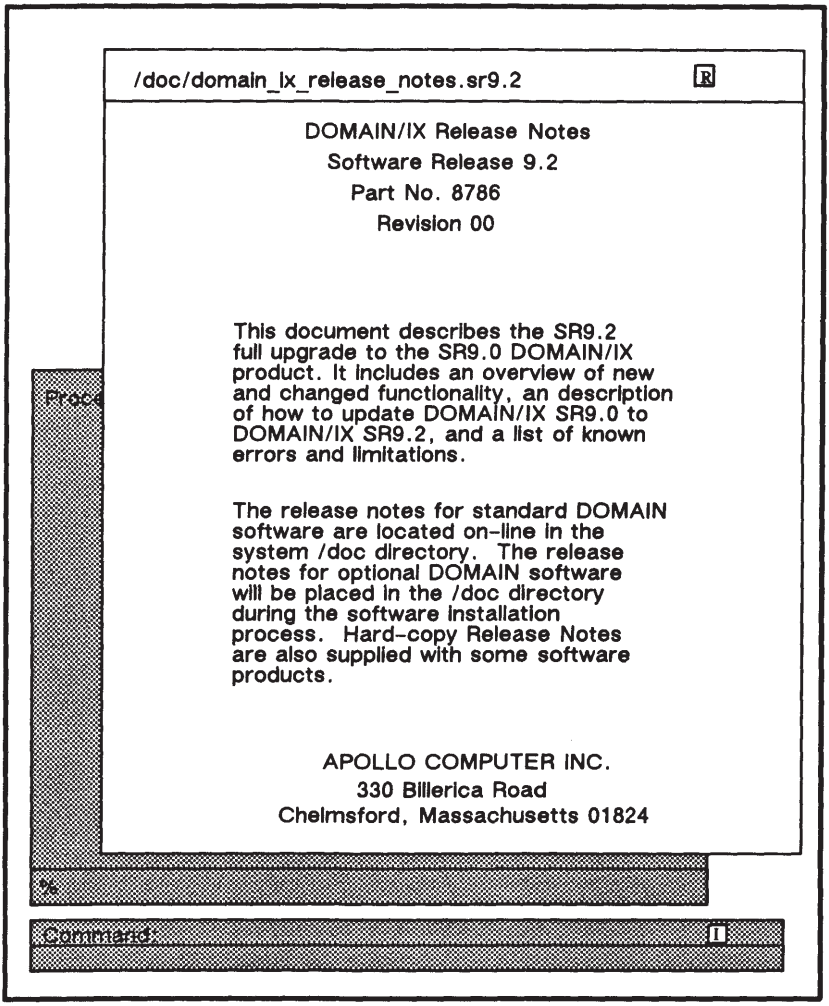


Figure 4–3. A Read–Only Window and Pad

For now, leave this read–only window and pad on your display. You will use it to learn about the keys discussed in the next several sections.

Moving a Pad Under a Window

The keys listed in Table 4–1 move a pad around under a window so that hidden text is in view.

Table 4–1. Keys that Move a Pad

Task	Predefined Key
Move to first character in a pad	CTRL/T
Move to last character in a pad	CTRL/B
Move pad by pages	↑ ↓
Move pad by lines	<SHIFT> / ↑ <SHIFT> / ↓
Move pad by characters	<SHIFT> / ← <SHIFT> / →

As you read about moving a pad with the keys discussed in the next three sections, try using them to move the pad containing the file "/doc/domain_ix_release_notes.sr9.2".

Moving to the Top and Bottom of a Pad

The CTRL/T sequence causes the DM to position the cursor at the top of the current pad. CTRL/B performs the complementary function of positioning the cursor at the bottom of the pad.

Let's move the cursor from the top of our sample file, "/doc/domain_ix_release_notes.sr9.2", to the bottom of the file. To do this, press CTRL/B.

The DM positions the cursor on the last character in "/doc/domain_ix_release_notes.sr9.2". Now press CTRL/T to move the cursor back to the first character in the file.

Moving (Scrolling) a Pad Vertically

The vertical scrolling keys ↑ and ↓ scroll a pad vertically under a window in units of half the height of the window.

The DM command **pp** (pad page) lets you modify the amount of text that the boxed arrow keys scroll. To set the number of pages to scroll, simply type the command followed by the number of pages that you want to scroll. For example, to scroll three pages, type this:

```
Command: pp 3 <RETURN>
```

NOTE: A "page" is defined as the smaller of either the number of lines that fit in the window, or the number of lines between the bottom of the window and the next form feed or frame.

You may also scroll a pad vertically by lines rather than by pages. The <SHIFT>/ ↑ and <SHIFT>/ ↓ sequences move the pad up and down one line, respectively. The cursor position does not change relative to the pad; the pad simply slides by under the window.

Moving (Scrolling) a Pad Horizontally

The ← and → keys scroll a pad left and right under a window in 10–character increments.

The left and right scrolling keys <SHIFT>/← and <SHIFT>/→ scroll a window over a pad in single character increments.

When you have finished experimenting with the keys that move a pad under a window, close the window and pad containing "/doc/domain_ix_release_notes.sr9.2". To do so, keep the cursor in the window and press <ABORT>.

Managing Windows

For the next few examples, you'll need to create several new Shell process windows. (Use <SHELL> to do this.) You are now ready to

learn how to shuffle and stack these windows, as well as how to change a window's size and position on the screen.

The predefined window control keys listed in Table 4–2 change the size, position, and characteristics of windows on the screen. In addition, all window control keys cause the DM to completely display the selected window if any part of it is hidden. Note that we show mouse key functions where appropriate.

Table 4–2. Predefined Window Control Keys

Task	Keyboard	Mouse
Shuffle windows	<POP>	Center Key
Copy text to process input window	<AGAIN>	––
Enlarge or reduce a window	<GROW>	Left Key
Move a window	<MOVE>	––

Pushing or Popping a Window

Pushing or popping windows allows you to display windows that are partially or completely hidden by other windows on your screen. Pressing <POP> pops a window to the top of the pile or pushes a window to the bottom of the pile of windows on the screen. These keys allow you to shuffle windows the way you would papers on your desk. Figure 4–4 illustrates how <POP> works.

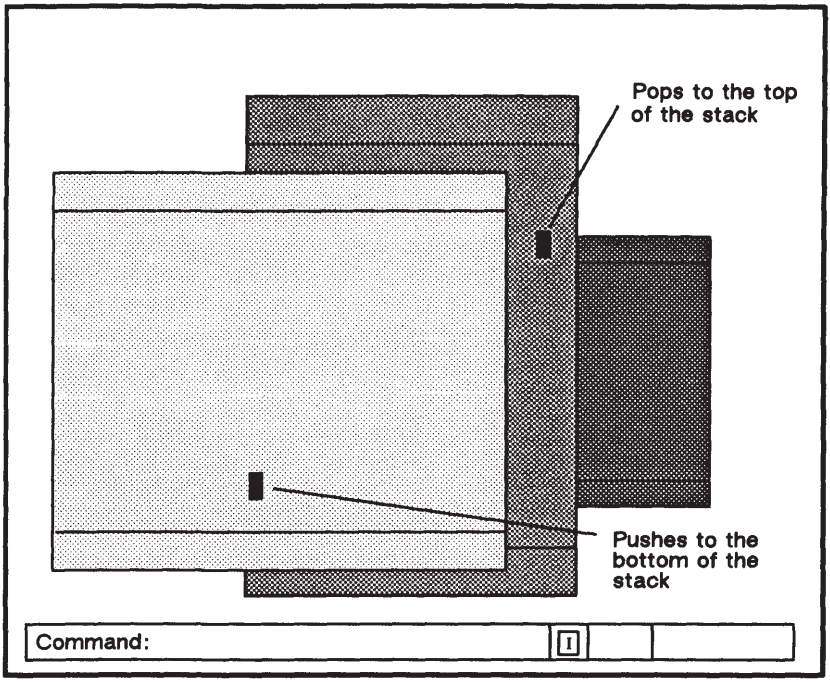


Figure 4–4. Pushing and Popping Windows

Try using <POP> to shuffle the windows on your screen. Move the cursor into a window that is partially hidden by another window. Press <POP>. The window pops to the top of the pile.

Now move the cursor into a window that is completely visible. Press <POP> again. This time the DM pushes the window to the bottom of the pile.

If you have a mouse, you can use its center key to push and pop windows.

Responding to Alarms

Experiment with <SHELL> and <POP> until you're comfortable manipulating windows. Now, position the windows so that one window overlaps the other's input window. Use <POP> to bring the hidden Shell process window to the top. Then use the **man** (on–line

manual page help) command to request information about the **cal** (print calendar) command. Type the following:

```
% man cal <RETURN>
```

Quickly press <POP> again and send the top window to the bottom of the stack.

If this exercise worked, the DM will display two small bells in its alarm window. If your system has a speaker, the DM also emits an alarm tone. The alarm informs you that the system is doing something in a window you can't see. Enter the DM command **ap** (alarm pop) to look at the window requesting your attention. For example, press <CMD> and type

```
Command: ap <RETURN>
```

After you press <RETURN>, the DM pops the window that needs attention. Refer to Chapter 4 in the *DOMAIN System User's Guide* for more information about alarms.

For now, leave the window containing the information on **cal** on your screen; we'll copy one of the sample commands from it in the next section.

Copying Text to the Process Input Window

Pressing <AGAIN> is an easy way to repeat a one–line Shell command that you typed earlier. This key copies a line from a transcript pad into the process input window.

To see how this works, copy the command line that you typed earlier (**man cal**). Follow these steps:

1. Move the cursor into the transcript pad, and position it on the first letter in "man".

2. Press <AGAIN>. The DM copies all the text from the current cursor position to the end of the line, and displays the copied text in the process input window.

If anything in the process input window is waiting for processing, the DM appends the copied line to that text. Once you have copied the command line to the process input window, you can edit the line or enter it just as it is by pressing <RETURN>.

Since the DM appends new information to the transcript pad, the command you want to copy may be hidden. Use the scrolling keys to move the transcript pad until the command is in view.

You can also copy text from other pads (pads displaying Help information or pads created with <READ> or <EDIT>, for example), and display the text in the process input window. You first mark the text you want to copy and then use <COPY> and <PASTE>.

Let's copy one of the sample command lines from the pad displaying the on–line help information about the cal command. If the window that contains the information you need is hidden, use <POP> to bring it to the top. Then follow these steps:

1. Position the cursor on the "c" shown in the first example (**cal 1985**) at the bottom of the window displaying the output from the **man** command.

2. Press <MARK> then ⇥ . The DM highlights the sample command.

3. Now press <COPY>.

4. Move the cursor into the process input window.

5. Press <PASTE>. The DM displays the copied command in the process input window.

6. Press ⇥ to move the cursor to the end of the pasted line.

After you press <RETURN> to enter the command, you'll see the calendar for the year 1985 displayed on the transcript pad.

Changing Window Size

If you want to make a window larger or smaller, use <GROW>. The DM lets you select a window edge or corner, which you can move across the screen to change the window's size. To change the size of one of the windows on your screen, follow these steps:

1. Decide which edge or corner you want to change, and move the cursor to it.

2. Press <GROW>. A flexible "rubberband" border appears. Figure 4–5 illustrates this flexible border.

3. Move the cursor to stretch or shrink the flexible border until it indicates the new window size you want.

4. Press <MARK>. The old window will shrink or expand according to the position of the flexible border.

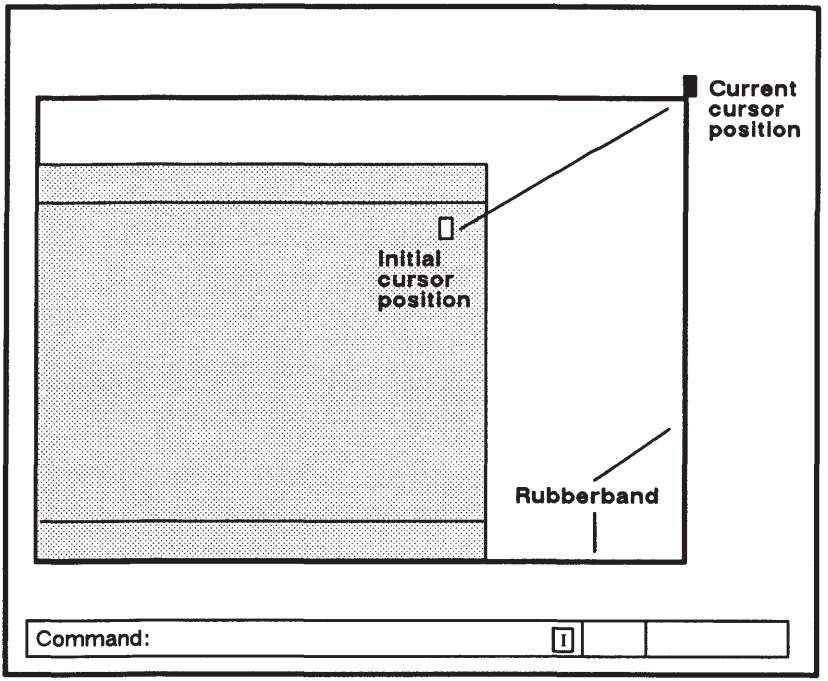


Figure 4–5. Changing a Window's Size

If you want to move just one edge, move the cursor only in the direction perpendicular to that edge. Moving the cursor vertically and horizontally causes a corner to move.

If, after the flexible border appears, you decide not to change the window size, press CTRL/X. Your node will emit a single "beep" and the DM will display the message "Echo mode aborted" on its output pad.

You can also change window size using a mouse. To do so, follow these steps:

1. Place the cursor near the edge or corner you wish to move.
2. Press the left–most mouse key and hold it down. The DM executes <GROW>, and the flexible border appears.
3. While you hold the left key down, move the mouse to indicate the new window size.
4. Now release the key. The DM executes <MARK>, and the window changes size.

Moving a Window

If you want to change the position of a window without changing its size, use <MOVE>. Use these steps to move one of the windows on your screen:

1. Place the cursor on an edge or corner of the window.
2. Press <MOVE>. A movable copy of the window border appears.
3. Move the cursor to position this border where you want the window to appear.
4. Press <MARK>. The window then moves to its new location.

If, after the movable border appears, you decide not to move the window, type CTRL/X.

Stopping a Process

When you've finished experimenting, you can stop the Shell processes you created and close the associated pads and windows.

Before you do, however, you may want to save the contents of the transcript pad. When you close a window or log off, the DM deletes all transcript pads on your display. However, the DM command **pn** (pad name) lets you save a copy of a transcript pad in a named file.

Move the cursor onto the transcript pad, then press <CMD> and type the following:

```
Command:  pn save_file <RETURN>
```

The pn command makes the temporary transcript pad a permanent file named “save_file”. The DM continues to add transcript pad output to “save_file” until you stop the process and close the windows. Refer to Chapter 4 in the *DOMAIN System User’s Guide* for more information about the pn command.

To stop a Shell process and close its windows and pads, follow these steps:

1. Move the cursor into the process input window and press CTRL/D. (To enter this *control key sequence*, hold down <CTRL> while you press <D>.) CTRL/D stops the process, deletes the input window, and closes all pads associated with the process. The system displays:

   ```
   % ***EOF***
   logout
   ***Pad Closed***
   ```

   NOTE: The “logout” message shown above only occurs in a C Shell. If using a Bourne Shell, you should still see an “EOF” (End–of–File) and a “Pad Closed” message when you execute a CTRL/D.

   The process input window disappears from your screen, but the window to the transcript pad remains.

   NOTE: If you have started other Shell processes from the same process window, you must execute a CTRL/D for each of these processes. Only after each of the processes started in that process window has been stopped will the “Pad Closed” message appear.

2. To remove the window to the transcript pad from your screen, keep the cursor within the window and press <EXIT>.

Suspending a Process

In addition to being able to stop a Shell process, you can suspend it for a period of time and resume processing later. This action does not close the windows and pads associated with the process.

To suspend a Shell process, move the cursor into the process input window and press CTRL/Z. To begin again where processing left off, move the cursor into the process input window and press CTRL/J.

NOTE: You can only suspend a process if your keyboard definitions have been redefined to “bsd4.2_keys” or “bsd4.2_keys”.

Summary

In this chapter, you learned how the DM controls windows, pads, and processes. The next chapter explains how the DOMAIN/IX system organizes information.

Chapter 5

Organizing Information

The DOMAIN/IX system organizes related information in *files*. We supply certain files with system software. You can create other files, and determine their contents. A file might contain a memo, a program, or a picture — anything you like.

The DOMAIN/IX system organizes related files in *directories*. You can also create directories and decide which files to store in them. We will explain how to create files and directories later in this chapter. The system keeps track of its files and directories by arranging them in a hierarchical structure called the *naming tree*.

This chapter introduces the basic concepts you'll need to understand and use the naming tree. Figure 5–1 shows a sample naming tree.

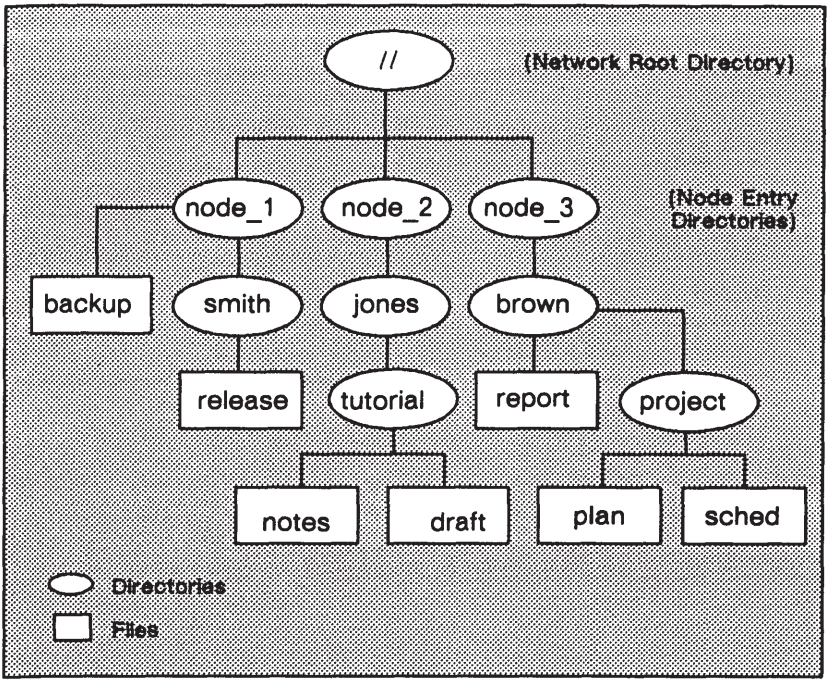


Figure 5–1. A Sample Naming Tree

The naming tree includes every file and directory in the network (not just on your node). Each directory in the naming tree appears above the files and subdirectories that it contains. Naming tree components are called *objects*.

In addition to files and directories, the naming tree includes a third type of object — *links*. A link points directly to, or contains the name of, another network object. When the DOMAIN/IX system uses a link, it may replace the *link name* with the *object name* that the link contains. We will explain how to create and use links later in this chapter. For now, just think of links as a special object type that enables you to take a detour from one part of the naming tree to another.

The object names that we use in the examples in this chapter are based on the sample naming tree in Figure 5–1. The object names in your naming tree will be different.

Using Pathnames

In order to use an object in the naming tree, we must be able to locate it. A *pathname* describes the path that the operating system must take to get from some starting point in the naming tree to a destination object. For example, consider the file “notes” in Figure 5–1. One pathname we could use to locate “notes” looks like this:

```
//node_2/jones/tutorial/notes
```

A pathname begins with the starting point’s name and includes every directory name between the starting point and the destination object. A pathname ends with the destination object’s name. Slashes separate names within a pathname. An individual name (between slashes) may not exceed 32 characters. The entire pathname may not exceed 256 characters, including the slashes.

In the pathname “//node_2/jones/tutorial/notes”, the starting point is the directory at the top of the naming tree, called the *network root directory*. (Double slashes (//) refer to the network root directory.) The *destination object* is the file “notes”. The directories between the starting point (//) and the destination object ("notes") are “node_2”, “jones”, and “tutorial”. Figure 5–2 highlights the path the system follows in its search for “notes”.

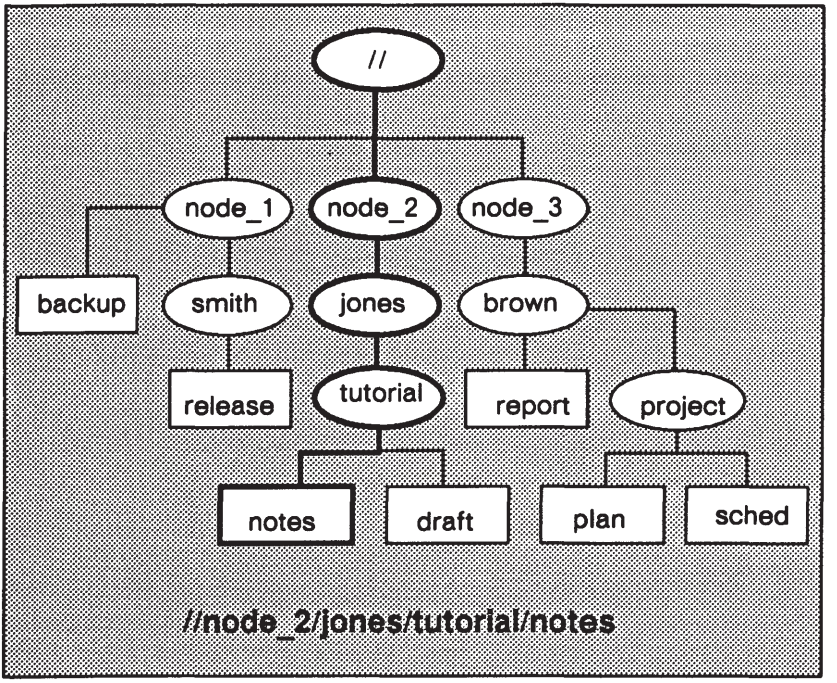


Figure 5–2. Sample Pathname

As you'll see in the following sections, not all pathnames begin with the network root directory. The system provides shorter ways to specify a pathname, depending on your current location in the naming tree.

Let's examine the parts of the naming tree more closely.

Where Am I?

Perhaps Figure 5–1 would be more meaningful if it included a "You Are Here" sign. Let's work our way from the top of the naming tree, the network root directory, to the directory you're currently using — your working directory.

The Network Root Directory (//)

The network's top–level directory, the *root directory* (//), is a list of directory names. It contains the name of each network node's top (or entry) directory. To display the contents of your network root directory, type the **ls** (list directory) command and the pathname // as follows:

```
%  ls // <RETURN>
```

Use the ls command to list the contents of any directory in your network. You can refer to any object in your network by specifying a pathname that begins with the root directory (//). (You must have the appropriate *permissions* to access the objects you specify. We explain these rights later on in this chapter.)

If you could list the contents of the root directory in our sample naming tree (Figure 5–1), you would see "node_1", "node_2", and "node_3" listed on your screen. Thus, the sample naming tree represents a network of these three nodes.

Your Node Entry Directory (/)

The *node entry directory* is the top directory on each node and is a subdirectory of the network root directory. In our sample naming tree, the node entry directories are "node_1", "node_2", and "node_3". To display your *user ID* and your node's entry directory name, type the **/com/lusr** (list user) command as follows:

```
%  /com/lusr –me <RETURN>
```

Note that the command line here uses the **–me** *command option*. (Most commands let you modify command execution by specifying one or more options. For more information about command options, see the next chapter.) The **/com/lusr –me** command line produces information in the following format:

In our example, the command output assumes that your node is "node_2", your user ID is "jones", your project identifier is "systest", your organizational identifier is "dev", and the identification number of the node that you are currently using is "24XP".

If you omit the **–me** option, the **/com/lusr** command lists all network users and their respective node entry directories. If you're using a *diskless node*, your node uses the entry directory of a disked partner node.

You can use a single slash (/) as a shorthand way of referring to your node's entry directory. For example, the following command lists the contents of your node entry directory:

```
% ls / <RETURN>
```

When you begin a pathname with a single slash (/), the system begins its search in the node entry directory of the node that is physically connected to your display unit. (If your node is diskless, the system begins its search in the entry directory of your node's disked partner.)

You could also list the contents of your node entry directory by typing the **ls** (list directory) command and a pathname that includes the root directory (//) and your node entry directory name. For example, if you were working on "node_1" in the sample naming tree shown in Figure 5–1, the command

```
% ls //node_1 <RETURN>
```

would list "backup" and "smith".

To refer to the entry directory on another node, use a pathname that starts with the root directory (//) and includes the name of the other node's entry directory.

Your Home Directory

The directory where the system locates you when you log in is called your *home directory*. Your user account contains your home directory name. When you log in, the system places you in the home directory specified in your user account.

You can go from any location in the network naming tree to your home directory by using the **cd** (change directory) command as follows:

```
% cd <RETURN>
```

Your Working Directory

Your *working directory* is the directory in which you are currently located. When you log in, the system sets the working directory to the home directory specified in your user account.

To display the name of the directory you're using (your working directory), enter the **pwd** (print working directory) command as follows:

```
% pwd <RETURN>
```

The working directory influences where a process creates or searches for object names (the names of files, directories, or links). For example, let's say your working directory is "//node_1/smith" in our sample naming tree. If you instruct the process to create a directory named "newdir" using die **mkdir** (make directory) command, the UNIX Shell process creates the new directory in "//node_1/smith" (the current working directory). For example:

```
% mkdir newdir <RETURN>
```

The new directory, "//node_1/smith/newdir" becomes a subdirectory of the current working directory, "//node_1/smith". The new directory appears beneath the current working directory in the naming tree. (If you enter the **ls** command, "newdir" appears in the list of the working directory contents.) Figure 5–3 illustrates the location of "newdir" in the naming tree.

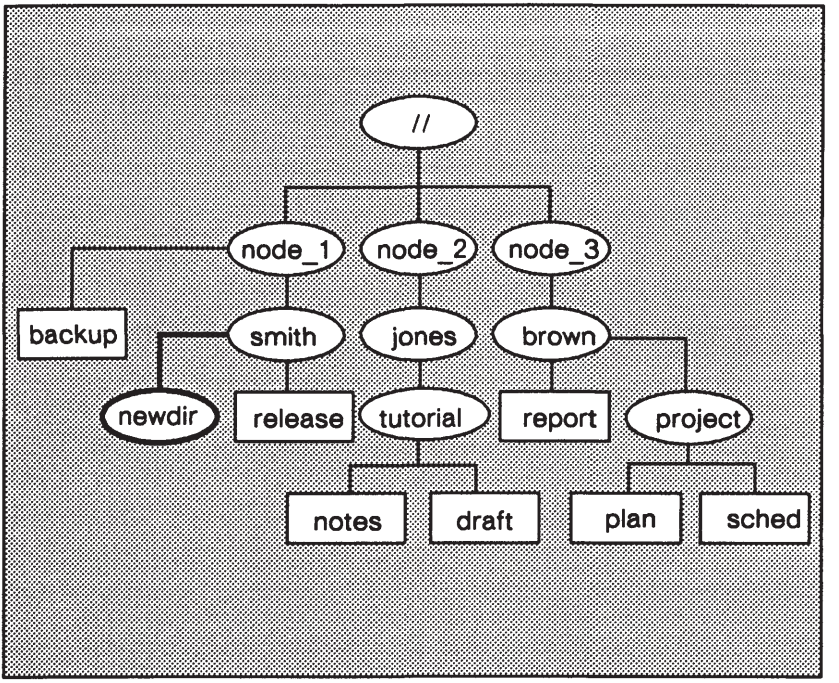


Figure 5–3. A New Directory in the Naming Tree

You can change the working directory to another directory using the **cd** command. Type

```
% cd new_directory <RETURN>
```

where "new_directory" is the name of the new working directory.

Parent Directories

A *parent directory* is the directory above the current working directory. For example, if your current working directory is "//node_1/smith/newdir", the parent directory is "//node_1/smith". To display the parent directory's name and contents, type

```
% ls .. <RETURN>
```

The double periods (..) refer to the directory one level above the current working directory. You can use a combination of double periods and slashes to cause the system to back up more than one level above the current working directory.

For example, let's say that your current working directory is "//node_1/smith/newdir/dir1/data", and you want to change the working directory to "//node_1/smith/newdir". You would type the following:

```
% cd ../..
```

Pathname Symbols — A Review

As you have seen in the previous sections, you can refer to any object in your network by using a pathname that begins with the root directory (//). The system also provides shorthand symbols that you can use in place of longer pathnames. Table 5–1 summarizes pathname starting–point symbols.

Table 5–1. Pathname Starting–Point Symbols

If your pathname starts with this symbol:	The system begins the name search in this directory:
//	Network root directory
/	Node entry directory
No symbol or .	Working directory
..	Parent directory

Figure 5–4 illustrates how you can use different symbols to refer to the same directory in the naming tree.

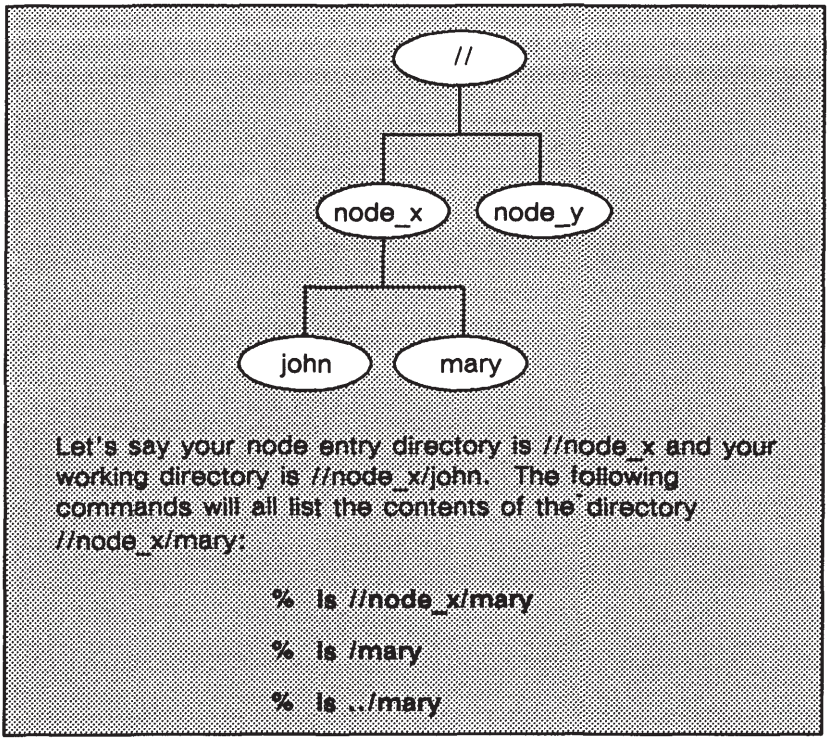


Figure 5–4. Pathnames Starting with //, /, and ../

Remember that the examples in Figure 5–4 assume that the node entry directory is "//node_x" and that the working directory is "//node_x/john". Each of the pathnames in Figure 5–4 searches for a destination object named "mary". The object's full pathname is "//node_x/mary". The pathname "/mary" instructs the system to look for "mary" in the node's entry directory (/). The pathname "../mary" instructs the system to move up to "//node_x" and then look for an object named "mary".

Using Links

As you use the system, you may find that you frequently access objects with very long pathnames. Instead of typing the long

pathname each time you want to use an object, you can create a special object type called a *link*.

Links are essentially of two types: *hard links* and *symbolic* (or *soft*) *links*. Hard links point directly to an object, and they may not span file systems or refer to directories. Symbolic links, however, point to link text (an object's pathname), can span file systems, and may refer to directories. Although you may want to use symbolic links more frequently, we'll discuss both types of links in this section.

Let's say that you work on "node_1" in our sample naming tree (Figure 5–1), and you often use the file "sched". To access "sched", you could type the following pathname each time you want to use the file:

```
//node_3/brown/project/sched
```

It would be much easier, however, to create a link to "sched". Then, whenever you wanted to access "sched", you could use the name of the link to get to it. To create a symbolic link, use the **ln** (link) command with the –s option. Let's call the link that represents the "sched" file in our example "mylink". To create "mylink" and place it in your current working directory, type the following:

```
%  ln –s //node_3/brown/project/sched mylink <RETURN>
```

This creates a symbolic link. A symbolic link contains the name of the file to which it is linked (the *link text*). Thus, when you use a link name as a pathname or as part of a pathname, the UNIX Shell substitutes the link text ("//node_3/brown/project/sched") for the link name ("mylink").

By default (with no options specified), the **ln** command creates hard links. Thus, if you type the following command line,

```
%  ln //node_3/brown/project/sched mylink <RETURN>
```

"mylink" points directly to "//node_3/brown/project/sched". Hard links are indistinguishable from the original directory entry to which they point, so any changes made to the file are independent of the name used to reference the file.

Therefore, if you create a hard link to "sched", and the file is subsequently modified using the DM editing function, your link points to the original "sched" file (called "sched.bak" after the

editing session is complete) and not the newly–modified version of the file (called "sched" after the editing session is complete).

NOTE: When you use the DM editor on a file, the modified version of the file assumes the original filename; the initial version of the file assumes the filename plus a .bak suffix.

If you would rather have your link always pointing to the latest version of "sched", you must create a symbolic link using **ln –s** as shown above.

For more information about links, see the **ln** command description in the *DOMAIN/IX Command Reference*.

File Permissions

As you saw in an earlier example, you can use the **ls** (list directory) command to determine the permissions set for any given file. Test this command on a file that you have created. The file's permissions should appear as some form of the set "rwxrwxrwx". Each "r" represents read permission, each "w" write permission, and each "x" execute permission. Note that, if a file is executable or writable, it is also automatically readable.

The first set describes the permissions of the file owner (usually, the user ID of the person who created the file). The next set shows group permission, allowing the creator of the file to restrict access to the file to a set project group (usually, the same as project ID). The final set shows permission of others (excluding file owner and project group).

NOTE: DOMAIN system software also provides another means of showing and setting file protections called an Access Control List (ACL). The AEGIS commands **/com/acl** and **/com/edacl**, respectively, perform these functions. Though you should use UNIX file protections most of the time, you should know that ACL format is more extensive (i.e., provides more types of permissions) than that provided by UNIX system permissions. See Chapter 8 of the *DOMAIN System Command Reference* for more information about ACL.

If you want to change permissions on a file that you own, use the **chmod** (change mode) command. You may change permissions of

the file according to absolute (using a hexadecimal number) or symbolic (using one or more alphabetic characters) modes and produce the same results.

NOTE: The owner of a file created via the DM editor may be listed as "<none>" until you explicitly specify the correct name of the file owner. To change ownership from "<none>" to your user ID, execute the **chmod** command on the file (see below). If you prefer, you may use AEGIS ACLs to name yourself, by default, as owner of files that you create within a directory.

As a brief example of how the **chmod** command is used, suppose that you want to change the permissions of a file that is currently readable, writable, and executable by all users on your network. You want to ensure that only you, the file owner, have complete access permissions. You also want to grant only read rights to your project group and all others. The following command line, where "myfile" is the file whose permissions are being changed, shows how to make the change using symbolic mode:

```
% chmod g=r,o=r myfile <RETURN>
```

This command line shows the same change done in absolute mode:

```
% chmod 744 myfile <RETURN>
```

For more information about specific modes to use for changing file permissions, consult the *DOMAIN/IX Command Reference*. Look under the entry for the **chmod** command.

Summary

This chapter described how the DOMAIN/IX system organizes files, directories, and links in the naming tree. You learned how to use pathnames to get from one part of the naming tree to another, and how to create new objects in the naming tree. The next chapter explains the many ways that you can create, edit, and read text files. It also describes how to print, copy, and delete files.

Chapter 6

Using Files

This chapter describes how to create, edit, and read files using DM function keys, control key sequences, and commands. It also mentions two other available editors, **vi** (a display–oriented editor) and **ed** (a line editor). In sample editing sessions, you'll create a new file, enter text, and modify the text. This chapter also explains how to print, copy, and delete files.

File Naming

Before you begin creating files, be aware of certain restrictions regarding their naming:

- Filenames are limited to 32 characters, and may include letters of the alphabet, numbers, dots (.), underscores (_), etc. In fact, the only character that they may not contain are

slash (/) or null. If a filename contains a character that the UNIX Shell interprets specially, that character must be escaped (i.e., preceded by a backslash).

- Text files should not be named with the same names as commands; otherwise, the Shell attempts to execute the text file as if it were a command.

- Case is an important consideration. All UNIX programs consider “file”, “File”, and “FILE” to be the names of three separate files. Remember that the DOMAIN/IX system is case–sensitive.

Using the DM to Create a File

The <EDIT> function key instructs the DM to open a window to a file. If you want to create a new file, you can use <EDIT> for that purpose. After you press <EDIT>, the cursor moves to the DM input window and the DM prompts you to specify the name of the file you wish to create (a file that does not already exist in your current working directory):

```
edit file:
```

The DM then opens an edit window for the file. Using what you learned about cursor control, you should be able to create a simple file. As you may recall, once you’ve positioned the cursor, you can use the following keys to correct errors:

- BACKSPACE
- INS
- CHAR DEL
- LINE DEL

When you’ve finished writing, leave the cursor in the window and press <EXIT>. This closes the edit window and saves your newly–created file.

If you decide that you do not want to save the file, use <ABORT> to discard it. When you press <ABORT>, the DM displays the following message:

```
File modified.  OK to quit?
```

Enter y or yes to discard the changes you've made and close the edit pad and window. The DM closes the edit pad and window without saving the changes you made during the session. Enter n or no to resume editing.

NOTE: When you create a file using the DM editor, UNIX programs may assign ownership of that file to "<none>" until you explicitly specify the correct name of the file owner. It is especially important to remember this when creating UNIX Shell start–up files, since UNIX Shells only read these files if they are owned by the person opening the Shell. This assignment of ownership to "<none>" takes place only because the DM editor is unable to determine who really owns the file. (To prevent further occurrences of this condition, you may wish to use AEGIS ACLs to specify yourself as owner of any files that are created within a particular directory. See the *DOMAIN System User's Guide* for more information on ACLs.) You can change ownership from M<none>'' to your user ID by executing the **chmod** command on the file.

As you create files, remember that if a file is executable or writable, it is also automatically readable.

Using the DM to Read a File

When you press <READ>, the cursor moves into the DM input window and the DM displays the prompt

```
read file:
```

You must specify a file that already exists. Since <READ> invokes the **cv** (create view) command, specifying a filename that does not exist causes the DM to issue the following error message:

```
(cv) filename – File not found
```

You can supply the name of a file in the current working directory, or use a pathname to refer to a file anywhere in the network.

Let's read a sample file. We've supplied a text file, called "sample_edit", that you can use to learn about the DM's read function. Press <READ> and type the file's pathname as follows:

```
read file:  /domain_examples/getting_started/sample_edit  <RETURN>
```

The DM displays the file "sample_edit". Notice the letter R (for Read) displayed in the window legend. Because "sample_edit" is now displayed in a read–only window, you cannot modify its contents. If you attempt to do so, the DM displays this message:

```
Text is read–only
```

NOTE: Although you cannot modify text displayed in a read window, you can change a read window into an edit window by pressing CTRL/M. Remember, however, that you must have write permission to the file you specify if you are to edit it.

The right–most mouse key also opens files for reading. To use this key, you must first display the name of the file you wish to read on your screen. As you recall, typing the **ls** (list directory) command at a UNIX Shell prompt lists all the names of the files in a given directory. When the name of the file you desire to read appears on your screen, point to it with the cursor. When you press the right–most mouse key, the DM displays the contents of the filename nearest the cursor.

When you finish reading "sample_edit", execute the following steps to close the file's window and pad:

1. Move the cursor into the file's window.
2. Press <ABORT>.

Using the DM to Edit a File

Now let's use <EDIT> to change the contents of an existing file. Press <EDIT>. As before, the DM asks you to specify the name of the file you wish to use (one that already exists). The cursor moves

into the DM input window, and the DM displays the following prompt

```
edit file:
```

After you enter the name of an existing file, the DM opens the file for editing. It also copies the original version of the file for backup purposes. This "backup" copy of the file appears in your directory with the original filename plus the .bak suffix.

Again, let's use "sample_edit" to learn about the DM's editing functions. To open "sample_edit", type the following pathname next to the DM edit prompt:

```
edit file:  /domain_examples/gettIng_started/sample_edIt  <RETURN>
```

When you press <RETURN>, the DM opens an edit window to the file "sample_edit" as shown in Figure 6–1. Notice that the pathname appears in the window legend. We've included a few errors (highlighted in Figure 6–1) that we'll use to demonstrate some editing functions.

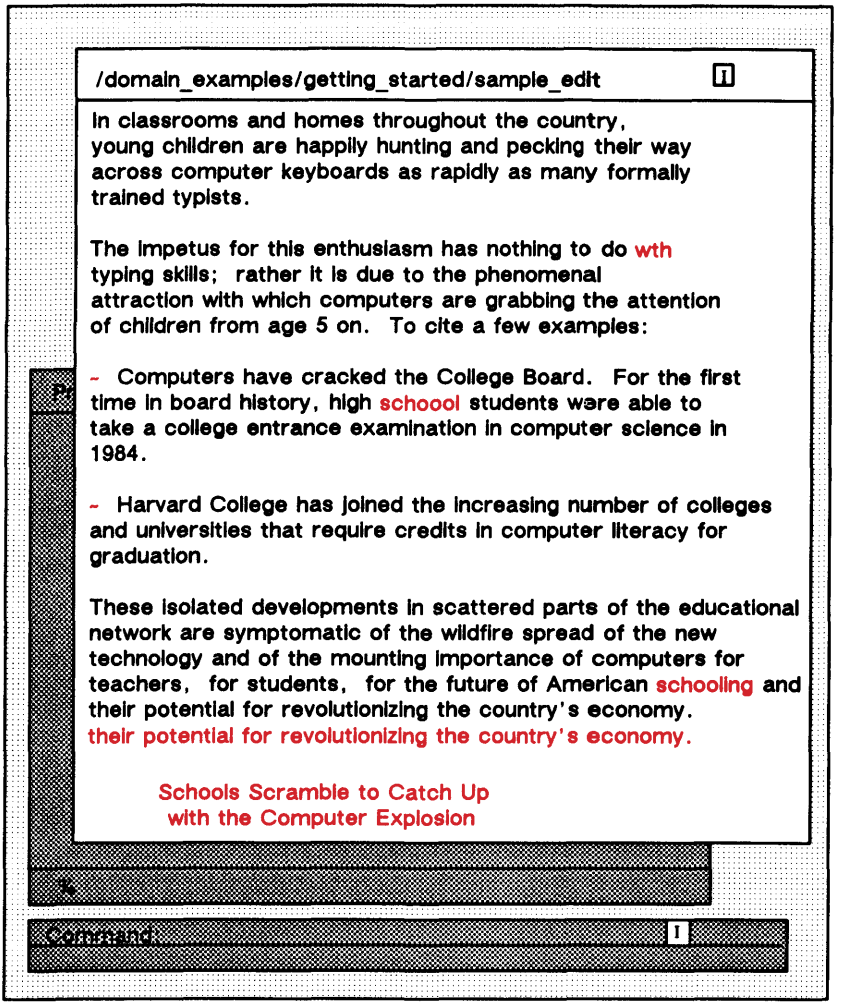


Figure 6–1. A Sample Editing Session Using the DM

Making Corrections to Text

You can move the cursor to any text position using the arrow keys, the touchpad, or the mouse. Once you've positioned the cursor, you can use <BACKSPACE>, <INS>, <CHAR DEL>, AND <LINE DEL> to correct errors. Although we have already mentioned how to

use these keys in a previous chapter, we will let you practice using them by working on a sample file.

Let's start by correcting the errors highlighted in Figure 6–1. We omitted the letter "i" from the word "with" in the first line of the second paragraph. Position the cursor on the letter "t" and type "i". Because the edit window is in insert mode, the letters to the right of the cursor move right when you insert a new character. By default, insert mode is on when you begin an editing session, and the letter I appears in the edit pad's window legend.

Press <INS> to turn insert mode off. Notice that the letter I disappears from the window legend. Now, any new characters that you type will overstrike (replace) existing characters. Let's try it. Move the cursor to the "s" in "schooling" in the last paragraph. Type the word "education". To turn insert mode back on, press <INS> again.

Move the cursor to an "o" in "schoool" in the second line of the third paragraph. To delete the extra "o", press <CHAR DEL>.

To delete the extra line in the last paragraph, move the cursor onto the line and press <LINE DEL>.

Defining a Range of Text

Before you can move, copy, or substitute text, you must define the range (or block) of text on which you want the DM to operate.

To define a range of text, place the cursor at the beginning of the range and press <MARK>. Then move the cursor to the end of the range. As you move the cursor, the DM highlights the text, showing you exactly what the range is. (Please note that the character under the cursor at the end of the range is not included within the range.) After you position the cursor at the end of the range, you either press a DM function key or type a command in the DM input window.

If you do not define a range, the default range for cut, copy, and substitute commands begins at the current cursor position and ends at the end of the line.

Cutting and Pasting (Moving) Text

The DM allows you to delete portions of text from a file, and to paste them in a different location in the same file, or in another file. You

must define a range of text before you execute the DM cut command.

To accomplish a cut or copy (see the next section) operation, the DM uses a paste buffer. A *paste buffer* is a temporary file that holds text that you have cut or copied so that it can be pasted in elsewhere.

Let's try a cut and paste operation. We'll move "Schools Scramble To Catch Up with the Computer Explosion" in Figure 6–1 from the bottom to the top of the file. Follow these steps:

1. To define the range for the operation, move the cursor to the first character position in the line before "Schools Scramble To Catch Up", and press <MARK>. Now move the cursor to the bottom of the file. As you move the cursor, notice the highlighting. This lets you see exactly what text falls within the range.

2. Press <CUT> (remember to hold <SHIFT> down at the same time). The DM deletes the text in the range you specified and writes it into the paste buffer.

   NOTE: The DM writes all deleted text into this buffer; however, it saves only the text deleted during the last DM operation. Therefore, don't delete anything else until you reinsert the paste buffer contents. Otherwise, you will lose the text you are attempting to move.

3. To place the contents of the paste buffer at the top of the file, press CTRL/T to move the cursor to the top of the file, and then press <PASTE>.

In this exercise, you used the DM's default paste buffer to hold the text you cut. You can create your own paste buffers (up to 100), each containing different blocks of text. To learn how to create and use paste buffers, see the cut, copy, and paste commands in the *DOMAIN System User's Guide*.

To delete a range of text without pasting it in elsewhere, define the range and then press <CUT>.

Copying and Pasting Text

The DM allows you to copy text into the paste buffer without deleting it from your file. The steps for copying text are similar to the steps for moving text. The difference is that you use <COPY> instead of <CUT>. You can copy text to a different location within a file or into another file.

In the next exercise, we'll copy a range of text into a new file. To do so, follow these steps:

1. Define any range of text in the file "sample_edit" in Figure 6–1. Move the cursor to the beginning of the range you want to copy, and press <MARK>, Then move the cursor to the end of the range. The DM highlights the text as you move the cursor.

2. Press <COPY>. The DM copies the range of text into the paste buffer. The original text remains undisturbed.

3. To create the new file, press <EDIT>. Next to the "edit file:" prompt in the DM input window, type

   edit file: copied_text <RETURN>

   The DM opens an edit window and pad to the new file "copied_text". The cursor is in the upper–left corner of the new file.

4. Press <PASTE>. The text you copied to the paste buffer appears in the file "copied_text".

To close "copied_text" and save its contents, keep the cursor in the window and press <EXIT>.

Searching for Text

The DM search function makes it easy for you to locate a particular text string in a file. In the DM input window, you type the string between slashes (/string/) or backslashes (\string\). A string enclosed in slashes instructs the DM to search forward from the current cursor position, and a string enclosed in backslashes instructs the DM to search backward from the current cursor position. For

example, to search for the string “computer” in “sample_edit”, follow these steps:

1. Press CTRL/T to move the cursor to the top of the file.
2. Press <CMD> to move the cursor to the DM input window.
3. Enter the search command as follows:

```
Command:  /computer/ <RETURN>
```

The DM then moves the cursor to the first occurrence of “computer” (in the first paragraph).

The CTRL/R and CTRL/U sequences repeat the last search forward or backward, respectively. For example, if you press CTRL/R now, the DM searches forward for the next occurrence of computer (in the second paragraph). If you then press CTRL/U, the DM moves the cursor back to the first occurrence of “computer” in the first paragraph.

Because the DM saves your most recent search command, you can repeat a search even if you’ve entered other (non–search) commands since you entered the search command.

Searches are case–insensitive by default. This means that “/mary/” locates “mary”, “MARY”, “Mary”, and even “mARY”. You can use the DM command **sc** (search case–sensitive) to perform case–sensitive searches. The *DOMAIN System User’s Guide* explains this particular command.

Canceling a Copy, Cut, or Search

To cancel a copy, cut, or search operation, use the CTRL/X sequence. For copy and cut operations, the DM removes the highlighting that indicates the range of text you are defining. For a search, the DM displays the message “Search aborted”.

Substituting Text

The DM also provides an easy–to–use search and substitute command, **s** (substitute). You can use it to search a file or part of a file for a text string, and to replace the string with a new string. First, you must define the range of text you wish to search and substitute.

Then, you can use the DM search and substitute command to perform the replacement.

For example, to replace every tilde (~) in "sample_edit" with the letter "o", you must:

1. Define the range of text you wish to search and substitute. The range for this search/substitute operation is from the top to the bottom of the file. To communicate this information to the DM:

   A. Move the cursor to the top of the file (CTRL/T does this) and press <MARK>.

   B. Next move the cursor to the bottom of the file (CTRL/B does this) and press <CMD>.

      Steps A and B tell the DM that the range for the next DM command begins at the top of the file and ends at the bottom.

2. Next, enter the following search and substitute command:

   Command: s/~/o/ <RETURN>

   The DM replaces every tilde (~) in the file with the letter "o". If you had not marked a range, the DM would have searched only the line that was to the right of the cursor when you pressed <CMD> in Step 1B.

Undoing Previous Commands

The <UNDO> function reverses the effect of the most recent DM command you entered. It keeps a history of activities in the DM input pad and edit pads in reverse chronological order. To undo the previous DM command, press <UNDO>. Successive use of the UNDO function key or command undoes DM commands further back in history.

The history of DM activities consists of circular lists (one for input pad activities and one for each edit pad). When the lists are full, they eliminate the oldest entries to make room for new ones.

NOTE: <UNDO> works for DM operations only; it does not reverse the effect of Shell commands once those commands have been executed.

Using the UNIX Display–Oriented Editor (vi)

You may prefer to use **vi**, the UNIX display–oriented (visual) text editor, when you are editing files. When you use **vi**, changes you make to the file are reflected in what appears on your node's screen. The position of the cursor on the screen indicates the position within the file. The editor uses commands that allow you to manipulate text in a variety of ways.

Consult the *DOMAIN/IX Text Processing Guide* for complete details on the use of **vi**. For a much briefer description of usage, see the *DOMAIN/IX Command Reference*, or simply execute the **man** command from a UNIX Shell for an on–line display of the command reference page.

Using the UNIX Line Editor (ed)

The standard UNIX line editor is named **ed**. When you specify the name of an existing file, **ed** reads the file into a temporary file called its *buffer*. Thus, the line editor operates on a copy of the file it is editing. Changes that you make in the copy have no effect until you write them.

Consult the *DOMAIN/IX Text Processing Guide* for complete details on the use of **ed**. For a much briefer description of usage, see the *DOMAIN/IX Command Reference*, or simply execute the **man** command from a UNIX Shell for an on–line display of the command reference page.

Copying a File

To copy the contents of a file and store it in another file, use the **cp** (copy file) command at the UNIX Shell prompt. Specify both the name of the file to be copied and the name you intend to give the new file. For example, to make a copy of "sample_edit" and store it

in a new file called “copied_file” in your current working directory, enter the following command:

```
% cp /domain_examples/getting_started/sample_edit copied_file <RETURN>
```

If you enter the **ls** command at the UNIX Shell prompt, you’ll see “copied_file” listed in the contents of your working directory.

Printing a File

In order to print a hard–copy of a file, your network must support one or more printing devices, and it must be capable of executing a process called a *Print Server*. When the Print Server is running, you can print a file using a print command or a special print menu. Remember, however, that you must have read permission on the file that you are attempting to print.

Using a Print Command

To produce a printout of a file via command, enter the **lpr** (*bsd4.2*) or **lp** (*sys5*) command and the pathname of the file to be printed. For example, the following *bsd4.2* version of the command prints three copies of the file “copied_file” on an available line printer (the default printer is site dependent):

```
% lpr –#3 copied_file <RETURN>
```

To print the file on a specific line printer, use the following command (where “spin” is the designated name of the printer to be used):

```
% lpr –P spin copied_file <RETURN>
```

The **lpr** command copies the file to a temporary file. This file resides on the node that controls your printer. Because **lpr** has submitted a copy of your file to the printer, you can edit the original version without changing the copy that is printing.

Ask your system administrator for the names of the printing devices at your installation.

Using a Print Menu

Instead of using a command to print your file, you may choose to use a print menu (see Figure 6–2). This menu lets you specify various arguments and options by typing information and making selections from the menu.

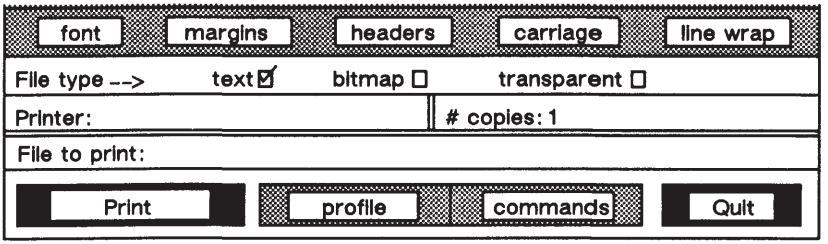


Figure 6–2. The Print Menu

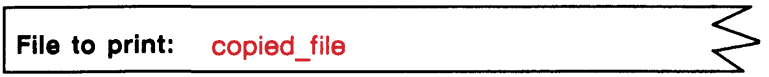
To display the print menu on your screen, enter the **/com/prfd** (print file dialog) command as follows:

```
%  /com/prfd <RETURN>
```

The **/com/prfd** command displays the print menu in a special window at the top of the Shell process transcript pad. An arrow cursor appears in the upper left corner of the menu.

NOTE: The arrow does not automatically become the typing cursor. You must press <F1> on your keyboard or the left key on your mouse in order for this action to take place.

To use the print menu to print the file "copied_file", move the arrow cursor to the space following the File to print: prompt. A small typing cursor appears on the line. Type "copied_file" as follows:



To print "copied_file" on another printer, move the arrow cursor to the space following the Printer: prompt.The typing cursor appears

on the line. Type the name of the printer you want to use. For example, to specify printer "spin", enter the following:

Note that the default number of copies (one) appears on the screen automatically. You may, of course, move the cursor to the copies window and change the number if you desire more than one copy of the file to be queued for printing.

When you're satisfied with the information that you've supplied on the print menu, move the arrow cursor to the box labeled Print. Press either the space bar or function key <F1>. If you're using a mouse, press the left mouse key. The following messages appear on the transcript pad:

```
Queuing data file "copied_file".
//node/owner/copied_file queued for printing.
File "/sys/print/queue/spin.owner.copied_file" queued  for printing.
```

This message means that the **/com/prfd** command has copied "copied_file" to a temporary file that has been named "spin.owner.copied_file". This file resides on the node that controls your printer in the directory "/sys/print/queue".

The print menu remains on your screen, allowing you to print additional files. When you're ready to exit from the print menu, move the arrow cursor to the box labeled "Quit". Press the space bar, <F1>, or if you're using a mouse, the left mouse key. The menu disappears from your screen, and the cursor reappears next to the Shell prompt.

Deleting a File

To delete files from directories, use the **rm** (remove file) command, and the name of the file to be removed, at the UNIX Shell prompt. For example:

```
%  rm copied_file <RETURN>
```

deletes the file "copied_file" from your current working directory. To delete a file anywhere else in the network, supply a pathname.

You must, however, have write permission to the file before you can delete it.

Summary

In this chapter, you learned how to create and read files. You also learned about various editors that are available under the DOMAIN/IX system. Furthermore, you became acquainted with methods for printing, copying, and deleting files using UNIX Shell commands. Chapter 7 contains more specific information about UNIX Shells.

Chapter 7

Using a UNIX Shell

As you saw earlier, DOMAIN/IX software lets you perform traditional computing activities, such as printing files and compiling and running programs. When you enter a command at a UNIX Shell prompt, the Shell invokes the utility program to perform the task you request. For example, when you entered the **rm** (remove file) command at the end of the last chapter, the UNIX Shell invoked the utility program that removes files.

The two UNIX Shells that we discuss in this chapter are the C Shell (supported by the *bsd4.2* version) and the Bourne Shell (supported by both *sys5* and *bsd4.2* versions). In particular, we describe:

- How a Shell command line is structured
- How the Shell interprets command lines and locates commands

- How to use symbols that have special meaning to the Shell
- How to create Shell scripts

Command Format

The simplest UNIX Shell command consists of a command name. For example:

```
%  ls <RETURN>
```

The ls (list directory) command lists the contents of your current working directory.

Using Command Arguments

Most Shell commands accept one or more *command arguments* to specify the object (file, directory, or link) upon which the command will act. For example:

```
%  ls  your_directory <RETURN>
```

In this example, "your_directory" is the argument. The **ls** command lists the contents of "your_directory". You must use one or more spaces to separate a command from its arguments, and to separate arguments from each other.

Using Command Options

In addition to command arguments, most Shell commands accept *command options* to modify the action of commands. They generally appear before command arguments. Most options are preceded by a hyphen, although this is not always the case. It is best to check the *DOMAIN/IX Command Reference*, or the on–line manual page (using the **man** command), for specific usage instructions in the case of each option. If an option is preceded by a hyphen, do not include any space between the hyphen and the option. For example:

```
%  ls –l <RETURN>
```

The **ls** command, used with the –1 option displays the contents of the current working directory in "long" format, supplying mode, number of links, owner, size in bytes, and time of last modification for each file in the directory.

Entering Multiple Commands on a Line

To enter more than one command on a single line, separate the commands with a semicolon (;).For example:

```
%  ls;date <RETURN>
```

This command lists the contents of the current working directory, then displays the date. Here's another example that you'll probably use often:

```
%  cd;ls <RETURN>
```

The **cd** (change directory) command changes the working directory to your home directory, and then the **ls** command lists its contents.

Command Line Processing

Most commands are files. When you press <RETURN> to enter one of these commands, the Shell searches for a file with the same name as the command you specified. If the Shell finds such a file, it loads and executes it as a program. Next, the Shell passes any arguments and options on your command line to the program with the same name as the command. After it executes the program, the Shell returns its prompt, which means that the Shell is ready for your next command. For example, when you enter the following command:

```
%  ls –l my_directory <RETURN>
```

the Shell searches for a file named "ls". The Shell then loads and executes the program that it finds in the file "ls". Next, the Shell passes the option and the argument to the program. After it executes the program and displays the contents of "my_directory" in a single column, the Shell redisplays a prompt on its input pad. You can then enter another Shell command.

Command Search Paths

When you enter a command that is the name of a file, the UNIX Shell takes a particular route (referred to as a *command search path*) in determining which directories to search for the command file. You can display the current search path of the UNIX Shell you're using by typing the **echo** command, with "$PATH" as its argument. For example:

```
% echo $PATH <RETURN>
```

NOTE: "PATH" is a Shell variable. By convention, Shell variables appear entirely in uppercase.

By default, when you have the *bsd4.2* version of DOMAIN/IX software installed on your node, the following three directory names are displayed when you enter the above command:

```
:/usr/ucb:/bin:/usr/bin
```

NOTE: The *sys5* version of DOMAIN/IX software uses ":/bin:/usr/bin" as its default search path.

But let's take a closer look at what the command search path is really composed of. The sequence of directory names separated by colons that you see above is stored in the Shell variable called "PATH". Assuming, as always, the *bsd4.2* version, the Shell looks for command files in this order:

1. Your current working directory (.)
2. The primary system command directory (/usr/ucb)
3. A secondary system command directory (/bin)
4. A third, less frequently used, system command directory (/usr/bin).

As soon as the Shell finds a filename that matches the command you specify, it attempts to execute the program. As we mentioned earlier, don't create text files with the same name as commands because the Shell attempts to execute the text file as if it were a command file.

To include other directories in your search path, use the **setenv** command from the C Shell prompt. For example, if you wanted to add the **/com** directory to the end of your default search path, you would type the following:

```
%  setenv PATH :/usr/ucb:/bin:/usr/bin:/com <RETURN>
```

This saves you the trouble of always remembering to type a "/com" in front of the **/com** enhancements that you have learned thus far.

NOTE: In the Bourne Shell (both *sys5* and *bsd4.2* versions), you can change the current search path by resetting the "PATH" Shell variable, augmenting the previous value with a new one. For example, typing "PATH=$PATH:/com" at the Bourne Shell prompt adds the **/com** directory to the end of your default search path.

Then, if you retyped the **echo** command with "$PATH" as its argument, you would see the following results:

```
PATH=:/bin:/usr/bin:/com
```

NOTE: In the C Shell, the "PATH" environment variable is predefined to a particular default when DOMAIN/IX software is installed on your node. You can still use the **echo** command with "$PATH" as its argument to find what the default setting is. To change your search path, we recommend that you eventually create a ".cshrc" file in your home directory, with the desired arguments to "PATH" specified via the **set** command built into the C Shell. Include the following command line in your ".cshrc" file, if you want to add the **/com** directory to your search path:

```
set PATH=(. /usr/ucb /bin /usr/bin /com)
```

Using Wildcards

Wildcards are special characters that you can use to represent one or more pathnames. For example, the following command line contains the * wildcard:

```
%  ls *.bak <RETURN>
```

The * wildcard matches every file that ends in ".bak" (backup versions of text files). Therefore, this command line lists all the files that end in ".bak" in your current working directory.

The next example lists all the objects (files, directories, and links) in "//node7" beginning with one of the letters a through z.

```
%  ls [a–z]* //node7 <RETURN>
```

The [...] wildcard, where the ellipses (...) represents specified characters, matches any of the enclosed characters. A pair of characters separated by a minus sign (–) matches any character lexically between the pair.

Most Shell commands that accept pathnames as arguments also accept wildcards. Wildcards are relatively easy to use, but there are more of them than are described here. Furthermore, the wildcards accepted by UNIX Shells sometimes have a different meaning in an AEGIS Shell – or their meanings differ between the Bourne Shell and the C Shell.

For this reason, do not use any wildcarding until you are sure that you understand fully how they operate within the particular type of UNIX Shell you are using. "Using the C Shell" and "An Introduction to the Bourne Shell" in the *DOMAIN/IX User's Guide* give the most complete information about wildcards.

Redirecting Input and Output

In the previous examples in this book, the Shell process read the commands you entered (input) from its input pad, and it wrote the responses to your commands (output) on its transcript pad. The next two sections explain a few examples of special characters that you can use to redirect input and output (I/O). For complete information about the special characters that redirect I/O, see "An Introduction to the Bourne Shell" and "Using the C Shell" in the *DOMAIN/IX User's Guide*.

Writing Output to a File

Some Shell commands write output to the transcript pad. For example, the following **cat** (catenate) command line with only a filename as an argument, directs its output to the transcript pad:

```
%  cat myfile <RETURN>
```

Here, **cat** catenates "myfile" and displays the output on the transcript pad.

To direct the output to a file instead of the transcript pad, use the greater than (**>**) symbol and another filename. For example:

```
%  cat myfile > myfile.out <RETURN>
```

The **cat** command first reads "myfile" and writes the catenated text to "myfile.out". The Shell creates the output file ("myfile.out") if it does not already exist. Now that the formatted text is stored in a file, you can read it or prepare it for printing.

Reading Input from a File

To instruct a Shell command to read input from a file instead of the input pad, use the less than (**<**) symbol and a filename. One example might be the **wc** (word count) command, which counts the number of characters (including blank spaces), words (character strings delimited by spaces, tabs, and new lines), and lines contained in the standard input. In this section, you'll learn how to instruct **wc** to read input from a file.

Before entering the **wc** command, create a text file to use as input to the command. To do so, follow these steps:

1. Press <EDIT> to create the file.
2. In the DM input window, enter the filename as follows:

   ```
   edit file:  sample_file <RETURN>
   ```

3. When the DM creates the edit window and pad on your screen, type the following lines of text:

```
wc counts the number of words,
characters, and lines contained in
the standard input (by default) or in
a file that you specify.
```

4. Press <EXIT> to close the edit window and pad.

Now instruct **wc** to get its input from the file "sample_file". Type the following command:

```
%  wc <sample_file <RETURN>
```

This tells **wc** to supply a count of the lines, words, and characters respectively contained in "sample_file". The "<sample_file" notation instructs **wc** to perform this count on the text in the file "sample_file". Then, **wc** writes its output to the transcript pad, like this:

```
4      23      129
```

The output represents lines (4), words (23), and characters (129) found in "sample_file".

Using Pipes and Filters

A pipe connects the standard output of one Shell command to the standard input of another. For example, suppose you wanted to find out how many words two of your files contain, when combined. You could do accomplish these tasks by typing the following two command lines (where "sample_file" is being created for the purposes of making a copy of the combined data in which a word count can be performed):

```
%  cat file1 file2 > sample_file   <RETURN>

%  wc –l < sample_file   <RETURN>
```

As you can see, this method is rather cumbersome. And, you probably don't really want to keep a copy of the union of the two files; all you really needed was a report on the total number of words

contained in the two files combined. If you use a pipeline, you can do both tasks in one simple operation, as shown below:

```
%  cat file1 file2 | wc <RETURN>
```

In this case, the **cat** (concatenation) command creates an output file only on the standard output and not in your working directory. The **wc** (word count) command uses this temporary data as standard input to produce the results on your display.

There is no limit on the size of a pipeline that you can create. You should, however, remember that pipes are interpreted by the Shell as being unidirectional.

Although you can't create a pipeline that has several programs feeding into one final program, you can build a type of "T–joint" in a pipe by using the tee command in your pipeline. This command tells the Shell to transcribe the standard input to the standard output and make copies in the named files.

The **tee** command is particularly useful for saving data being transmitted through the pipeline, data that is essentially written over in the process. For example, if you wanted to check a file called "test" for spelling errors, create a file named "test.errors" that contains the list of mispelled words found in "test", and create a separate file named "test.num" that provides only a count of the total number of mispellings in "test", you would type the following command line:

```
%  spell test | tee test.errors | wc –w > test.num <RETURN>
```

Since **tee** copies its input to the specified file(s) and to its output, the intermediate output from the execution of **spell** is saved in "test.errors", a copy of this data passes through the **wc** (word count) program, and the result of the execution of **wc** is written onto a new file named "test.num".

A *filter* reads its input, performs some specified operation on it, and then prints the result as output. An example of one useful filter is **grep**, which searches its input for lines that contain a string you have specified. For instance, the command line

```
%  ls | grep bak  <RETURN>
```

prints all lines from the output of **ls** that contain the string "bak".

Creating Shell Scripts

A *Shell script* is a file that contains a list of Shell commands. If you find yourself repeating a sequence of commands over and over again, create a Shell script containing the commands. You can then execute the entire command sequence with a single command.

To create a script, use <EDIT> to create the file, type the Shell commands, one per line, then press <EXIT> to close the file. To execute the script, type the script's filename in the process input window next to the UNIX Shell prompt. Be sure to read the remainder of this section for important details about Shell script execution.

Let's try an example. Suppose you wanted to change your current directory to your home directory, and then get a long listing of its contents (using the **–l** option with **ls**). You could accomplish these tasks by entering these Shell commands, one after another:

```
% cd <RETURN>

% ls –l <RETURN>
```

The **cd** (change directory) command sets the working directory to your home directory. The **ls** command displays the name of each object in your home directory. The **–l** option displays each object's mode, owner, size in bytes, etc.

If you create a Shell script containing the **cd** and **ls –l** command lines, you can execute both commands by simply entering the filename of the script. To create a script called "dir", follow these steps:

1. Get into your home directory by executing a **cd** at the UNIX Shell prompt.

2. Create a personal directory in which your Shell scripts can exist. Typically, your built–in scripts should reside in a "bin" subdirectory of your home directory. You already have a "/bin" subdirectory in your node entry directory, and the UNIX Shell uses this subdirectory in its command search path. Your personal "bin" should also be added to your search path so that the Shell looks there for commands as well. (See Chapter 5 for more information about adding

directories to your command search path.) Get into your "bin" subdirectory by typing

```
% cd bin <RETURN>
```

NOTE: Now, if you type a **pwd** (print working directory) command at the UNIX Shell prompt, you should see your node name, followed by your user ID, and finally, the name "bin".

3. Press <EDIT> to create the file that will contain your script. In the DM input window, enter the filename as follows:

```
edit file: dir <RETURN>
```

4. When the DM creates the edit window and pad, type the Shell commands, one per line, like this:

```
cd my_dir <RETURN>

ls –l <RETURN>
```

NOTE: By default, UNIX Shell scripts are interpreted by the Bourne Shell when no specific Shell is named in the first (comment) line of the script file. Therefore, if you have the *bsd4.2* version of the DOMAIN/IX system installed on your node, and wish to use the C Shell in your script, you must specify it by typing the following on the first line of the file:

```
#! /bin/csh
```

5. Press <EXIT> to close the edit window and pad.

Before you can execute your Shell script, you must make sure that the file is executable. By default, the files that you create with the DM editor are executable. As you remember, however, the owner of such files will be listed as "root". Therefore, you must use **chmod** on these files to make yourself owner of them. (See Chapter 5 for more information on this.)

Using the script that you just created, you can now execute the **cd** and **ls** command lines by typing the following:

```
% dir <RETURN>
```

Remember that, once you have included your personal "/bin" in your home directory, you can execute your Shell scripts from anywhere on the naming tree. You can use your Shell scripts in a variety of ways (e.g., as parts of a pipeline). When you're ready to create more sophisticated scripts, read "Using the C Shell" or "An Introduction to the Bourne Shell" in the *DOMAIN/IX User's Guide*.

Summary

In this chapter, you learned more about UNIX Shells. You read about how a Shell processes your commands and how it locates the command files you request. We also introduced you to some of the more powerful features of UNIX Shells, namely wildcards, symbols that redirect input and output, pipes and filters, and Shell scripts.

Now that you're familiar with the UNIX Shells, read the *DOMAIN/IX User's Guide* to learn how to perform specific tasks using either the C Shell or the Bourne Shell. See the *DOMAIN/IX Command Reference* for a brief explanation of how each Shell works, or use **man** to view information on the C Shell (**csh**) or the Bourne Shell (**sh**).

Chapter 8

Using Both DOMAIN/IX Versions

The two versions of the UNIX operating system that DOMAIN/IX software supports provide a variety of similar (though seldom identical) system services through kernel and library functions. While a particular function may exist in both the *sys5* and the *bsd4.2* versions, the semantics of the function, and sometimes its arguments, may be subtly different.

UNIX programs traditionally assume that the runtime environment will be UNIX software of a certain lineage (AT&T or Berkeley) or even a specific version (AT&T System V or 4.2 BSD). DOMAIN/IX uses this information to create the foundation for its multiple–version support.

Even though you have both versions available to you at your site, you must select the one for which your operating environment (e.g., the

commands that are available with the particular version) will be targeted. The actual version selector is an *environment variable* called "SYSTYPE". We recommend that, as a beginning user of DOMAIN/IX, you ask your system administrator to set this variable for you.

Name Space Support

Since its formation, the UNIX file system has contained a few system directories with rather well–known names. These directories are: /usr, /bin, /etc, /dev, and /tmp. Table 8–1 provides a description of the basic contents of these directories (and some of their important subdirectories).

Table 8–1. DOMAIN/IX System Directories

Name	Object Type	Description
/bsd4.2	directory	top–level directory containing /usr, /bin, and /etc
/bin	symbolic link	basic programs in executable form (“binaries”)
/etc	symbolic link	administrative commands and database files used system–wide
/dev	normal link	device files
/tmp	normal link	temporary working space for files created during program execution
/usr	symbolic link	user file system with special directories for:
		system administration data (/usr/adm)
		user–oriented utilities (/usr/bin)
		spelling dictionary (/usr/dict)
		tutorial documents (/usr/doc)
		standard C include files that describe the standard system configuration for C programs (/usr/include)
		object libraries, data files, and auxiliary programs used by software development utilities (/usr/lib)
		on–line manual entries (/usr/man)
		misc. publication (font) files (/usr/pub)
		editor temporary files (/usr/preserve)
		working directories for communications programs (/usr/spool)
		binaries of programs developed at UC–Berkeley (/usr/ucb)

The structure and content of each of the directories differs within the two UNIX versions supported by the DOMAIN/IX system. To support identically named versions of these directories on the same DOMAIN file system, we have introduced *symbolic links*. Unlike regular links, symbolic links allow a portion of the link text to be replaced by an environment variable.

Symbolic links placed in your node's root directory during the installation procedure allow programs to use either the *sys5* or the *bsd4.2* versions of the "/bin", "/etc", and "/usr" directories (the other two directories mentioned above are common to both versions). Although the links to "/bin"," /usr", and "/etc" are normally created by the installation script, you may at some point need to create or re–create such links yourself using the **ln –s** (create symbolic link) command. For example, to create a SYSTYPE–dependent link for "/bin", use the following:

```
% ln –s /bin '$(systype)/bin'
```

NOTE: The single quotes around the link text are required. Otherwise, the dollar sign will be interpreted as a Shell metacharacter.

The SYSTYPE environment variable determines the selection of UNIX commands, libraries, spool directories, and the like. (Table 8–1 shows the top–level DOMAIN/IX directory organization.)

Environment Switching

The DOMAIN/IX system allows you to execute a *sys5* program from a *bsd4.2* UNIX Shell (or vice versa) without any knowledge of the UNIX version for which the program was targeted. When you invoke a program that is stamped with a specific version of the UNIX operating system (i.e., either *sys5* or *bsd4.2*), the SYSTYPE environment variable for the process in which the program is running is set to that version. The program "/etc/systype" displays the version stamp.

A Shell's SYSTYPE value defines the version of system directories that are searched when a command name is given. Thus, it defines the version of the commands that you execute. To simplify the execution of a particular version of a UNIX command from a differing version of a UNIX Shell, you can use the **ver** (display or set

version) command. The **ver** command comes in handy, for example, when you want to invoke a System V command from a BSD4.2 Bourne Shell, or even a C Shell. You can use **ver** in the following three ways:

- To display the current value of SYSTYPE. For example, if your SYSTYPE is set to "bsd4.2", typing the command without arguments, produces these results:

```
% ver

bsd4.2
```

- To change the current value of SYSTYPE to another value, thereby changing the version of subsequently executed commands. For instance, if your SYSTYPE is set to *bsd4.2* and you want to set it to *sys5*, type the following:

```
% ver sys5
```

  Now, you can verify that the version was indeed changed by invoking the **ver** command without any arguments, as in the first example above.

- To execute a specific version (*sys5* or *bsd4.2*) of a command without changing the current setting of SYSTYPE. For example, to execute the System V version of the **diff** command (which compares two large files and finds differing lines) from a BSD4.2 C Shell without changing the Shell to a System V Bourne Shell, type the following:

```
% ver sys5 diff file1 file2
```

  In this example, "file1" and "file2" are existing files in your current working directory that you wish to compare.

Summary

In this chapter, you learned about support for multiple versions of DOMAIN/IX software. Now that you're familiar with the DOMAIN/IX user environment and the functions of UNIX Shells and the Display Manager software, continue with the *DOMAIN/IX User's Guide* to learn specific information about these features.

Appendix A

Keyboard Control Summary

The following summarizes the functions of the keys described in this book. We have included information on both the low–profile keyboard (the one used in all our examples) and the older model 880 keyboard.

MOVING THE CURSOR

Task	Low–Profile Keyboard	880 Keyboard
Move left one character	←	←
Move right one character	→	→
Move up one line	↑	↑
Move down one line	↓	↓
Move to start of next line	CTRL/K	CTRL/K
Move to beginning of line	⇤	⇤
Move to end of line	⇥	⇥
Tab right	<TAB>	<TAB>
Tab left	CTRL/<TAB>	CTRL/<TAB>
Move to DM input window	<CMD>	<CMD>
Move to next window on screen	<NEXT WNDW>	<NEXT WNDW>

CREATING AND CONTROLLING A PROCESS

Task	Low–Profile Keyboard	880 Keyboard
Create a new process, transcript pad, and window	<SHELL>	<SHELL>
Stop a process, delete the input window, and close pads	CTRL/D	CTRL/D
Suspend a process (bsd4.2 only)	CTRL/Z	CTRL/Z
Restart a process after it has been suspended (bsd4.2 only)	CTRL/J	CTRL/J
Save the transcript pad in a file	<CMD> pn pathname	<CMD> pn pathname

MOVING A PAD UNDER A WINDOW

Task	Low–Profile Keyboard	880 Keyboard
Move cursor to first character in pad	CTRL/T	CTRL/T
Move cursor to last character in pad	CTRL/B	CTRL/B
Move pad by pages	↑ ↓	↑ ↓
Move pad by lines	<SHIFT>/ ↑ <SHIFT>/ ↓	<F2>, <F3> <F2>, <F3>
Move pad by characters	←, <SHIFT>/ ← →, <SHIFT>/ →	← →

MANAGING WINDOWS

Task	Low–Profile Keyboard	880 Keyboard
Change a window's size	<GROW>	CTRL/G
Move a window	<MOVE>	CTRL/W
Cancel changing window size	CTRL/X	CTRL/X
Cancel moving a window	CTRL/X	CTRL/X
Close pad and window; update file	<EXIT>	CTRL/Y
Close pad and window; no file update	<ABORT>	CTRL/N
Copy text to process input window	<AGAIN>	<F8>

MANAGING THE DISPLAY

Task	Low–Profile Keyboard	880 Keyboard
Request help on DM or AEGIS commands	<HELP>	See UNIX Shell command **help**
Request help on UNIX commands	See UNIX Shell command **man**	See UNIX Shell command **man**
Log in	<CMD> user ID	<CMD> user ID
Log out	<CMD> lo	<CMD> lo
Place a mark	<MARK>	<MARK>
Respond to DM alarm and pop window	<CMD> ap	<CMD> ap

CREATING A PAD AND WINDOW TO READ AND EDIT FILES

Task	Low–Profile Keyboard	880 Keyboard
Create edit pad and window through which to view it	<EDIT>	<EDIT>
Create read–only pad and window	<READ>	<READ>

EDITING A PAD

Task	Low–Profile Keyboard	880 Keyboard
Set read/write mode	CTRL/M	CTRL/M
Set insert/overstrike mode	<INS>	<INS MODE>
Insert newline character	<RETURN>	<RETURN>
Insert new line after current line	<F1>	<F1>
Insert end–of–file mark	CTRL/D	CTRL/D
Delete character at cursor	<CHAR DEL>	<CHAR DEL>
Delete character before cursor	<BACKSPACE>	<BACKSPACE>
Delete word of text	<F6>	<F6>
Delete from cursor to end of line	<F7>	<F7>
Delete entire line	<LINE DEL>	<LINE DEL>
Copy defined range of text to paste buffer	<COPY>	CTRL/C
Cut (delete) defined range of text and write it to paste buffer	<CUT>	CTRL/E
Paste (write) text in paste buffer into pad	<PASTE>	CTRL/O
Search forward for string	<CMD> /string/	<CMD> /string/
Repeat last forward search	CTRL/R	CTRL/R
Search backward for string	<CMD> \string\	<CMD> \string\
Repeat last backward search	CTRL/U	CTRL/U
Cancel copy, cut, or search	CTRL/X	CTRL/X

Task	Low–Profile Keyboard	880 Keyboard
Undo previous command	<UNDO>	<CMD> undo
Update edit file without closing edit pad	<SAVE>	<CMD> pw
Substitute “strIng2” for all occurrences of “string 1” in defined range	<CMD> s/string1/string2/	
Substitute “string2” for first occurrence of “string 1” in each line of defined range	<CMD> so/string1/string2/	

Appendix B

DM Environment Variable Settings

The following describes the DM environment variable settings that must be present in order for the examples in this book to work. Some of these variables are preset by the system; others have special significance to system software or other special attributes. We refer to environment variables that are defined by the system at log–in time as “preset”. The subset of these that you cannot delete or modify are referred to as “privileged”.

We do not recommend that you try to set the values for these variables yourself. If the examples in this book do not work as we have shown, please contact your system administrator. Information in this appendix should help you and your system administrator verify the accuracy of the settings on your node.

Variable Name	Description	Example
NODETYPE (privileged)	Type of node on which the process is running.	DN300
USER (privileged)	User's log–in name.	bill
LOGNAME (privileged)	Same as USER.	bill
PROJECT (privileged)	Project (group) ID under which the user logged in.	systest
ORGANIZATION (privileged)	Organization ID under which the user logged in.	devmt
NODEID (privileged)	Unique node identifier for the node on which the process is running, expressed in hexadecimal.	54cd
HOME (privileged)	User's home directory path–name, established at log–in.	//node1/bill
TERM (preset)	Device name of the "terminal" in use.	apollo_19l
TZ (preset)	Timezone string, where the first three characters are the standard timezone name, the middle number is the difference in hours between the standard timezone and UTC, and the last three characters are the daylight timezone name.	EST5EDT

Variable Name	Description	Example
UNIXLOGIN	Specifies that a UNIX log–in sequence is to be used in place of the standard DOMAIN log–in sequence. (Valid values are “true” and “false”.)	true
SYSTYPE	The UNIX system version in use.	bsd4.2
NAMECHARS	Specifies a set of characters to which special semantics are attached during name translation.	
PATH	Command search path to be followed. Dependent on setting of SYSTYPE.	:/bin:/usr/bin:/usr/ucb
SHELL	Type of Shell to be created by default. Dependent on setting of SYSTYPE.	/bin/csh

Here is an excerpt from a script used to set up the DM environment described in this book. Similar lines should appear as a part of the startup–file that sets up the initial operating environment on your node. Notice the setup of environment variables previously mentioned in this index. Also note the setup of the "/etc.rc" script (the **/com/cps** command creates a process server, of which "/etc.rc" is one type).

```
#  Select a default UNIX version for the node:
#
env SYSTYPE 'bsd4.2'

#  Use a UNIX login sequence:
#
env UNIXLOGIN 'true'

#  Run 'node_data/etc.rc (a Shell script that usually starts various
#  UNIX daemons, such ones used for the line printer facility, when
the node is booted):
#
cps /etc/run_rc
```

Appendix C

Command Synopsis

The following is a brief synopsis of some useful commands that the DOMAIN/IX system provides. In most cases, a standard UNIX command appears first on a line, followed by an available DOMAIN/IX extension. The *DOMAIN/IX Command Reference* and the *DOMAIN/IX Programmer Reference* materials explain the UNIX commands in more detail. The **/com** extensions are discussed in the *DOMAIN System User's Guide*.

access control

show default permissions	**umask** or **/com/acl** (with the –all option)
show current permissions	**ls** (when used with –l option), or **/com/acl**
change protection	**chmod** or **/com/edacl**

directory control

compare two directories	**diff**
delete a directory	**rm** (when used with –r option)
list directory contents	**ls**
create a directory	**mkdir**
move up one directory	**cd**
move up two directories	**cd** ../..
remove directory (empty)	**rmdir**
set working directory	**cd** *x* (where *x* is the target working directory
set to network root	**cd** //
show home directory	**echo** $HOME **printenv** HOME (*bsd4.2* only)
show working directory	**pwd**

file control

concatenate files	**cat**
copy a file	**cp**
copy std input to std output and named files	**tee**
create/modify magtape descriptor files	**/com/edmtdesc**
delete a file	**rm**
device descriptor file	**mknod**
lock a file	**/com/lkob**
maintain an archive	**ar**
rename a file	**mv**
unlock a file	**/com/ulkob**

link control

create a link	**ln**
delete a link	**rm**

network control

change group ID	**chgrp**
change file ownership	**chown** (*sys5* only) **/com/edacl**
list users logged in	**who** (with the –a option) or **/com/lusr**

list your login name	**whoami** (bsd4.2 only) **who** (with the am i argument) **/com/lusr** (with the –me option)

Appendix D

Error Message Summary

The following is a list of common error messages that you may encounter if you make input mistakes during your DOMAIN/IX session. Although it is not exhaustive, the list should help you to understand the meaning of several of the more cryptic responses that you may get from the programs you run.

Common Error Messages

Error Message	Interpretation
can't open file	File does not exist, or you do not have read permission. If you are sure that this does not apply, check to see whether you might be trying to access it from the wrong directory.
name : command not found	Command does not exist within any directories in your search path.
usage: command file1 file2	Command was not used properly. Usage, as it should be, is shown in message.
file: cannot create	File already exists and you have no write permission to it, or the directory that contains the file does not allow write permission (i.e., you cannot create any new files within that directory).
file: permission denied	See "cannot create" message (above).
file: text file busy	You are attempting to write to a file that is already locked.

Common Error Messages (continued)

Error Message	Interpretation
broken pipe	A command within the pipeline cannot pass the specified data to the next command, or it has not received input from the previous command.
bad directory	Attempted to use a file as a directory.
can't access file	File does not exist or you do not have read permission.
file: cannot unlink	You have tried to remove a file for which you have no permission to do so. This could also mean that the directory in which the file resides has restricted it from being deleted by permissions, thus protecting others.
file: 700 mode	The permission mode of the file is set so that you cannot perform the desired function. The "700" mode (or any other three–digit number that appears with this message) refers to the absolute mode given as the first argument to the **chmod** command. In this case, 700 mode means that the file is readable, writable, and executable only by its owner.

Glossary

Access Rights — These rights list the people who can use each object in the network, and specify how each person can use the object. UNIX access rights consist of read, write, and execute permissions.

Alarm Window — The Display Manager alarm window appears near the bottom of your screen. It displays a small pair of bells when a process displays a message in an output window that is hidden by an overlapping window.

Argument — See *Command Argument*.

Background Process — A non–interactive process that runs immune to quit and interrupt signals issued from your node. This allows you to start a task and then go on to another task while the system continues with the initial one.

BSD4.2 — The version of the DOMAIN/IX system that implements 4.2 BSD UNIX from the University of California at Berkeley.

Command — An instruction you give a program.

Command Argument — A command option or the name of the object upon which the command will act. Command arguments follow commands on the same line, although not all commands require an argument. (Also see *Command Option*.)

Command Option — Information you provide on a command line to indicate the type of action you want the command to take. (Also see *Default*.)

Command Search Path — The route that the Shell takes in searching through various directories for command files. A default search path exists for each of the DOMAIN/IX versions. You may add other directories of executable files which the Shell will then look through on its way to finding a particular command name.

Control Character — A special invisible character that controls some portion of the input and output of the programs you run on your node. (Also see *Control Key Sequence*.)

Control Key Sequence — A keystroke combination consisting of <CTRL> followed by another key. These combinations provide you with a shorthand method of specifying commands. To enter a control key sequence, hold <CTRL> down while you press another key.

Current Directory (.) — Also known as your *working directory*, this is the location in the hierarchical naming tree of the directory that you are working in at any given time. Entering the UNIX command **pwd** prints the name of your current directory.

Cursor — The small, blinking box initially displayed in the screen's lower left corner. The cursor marks your current typing position on the screen and indicates which program (Shell or DM) receives your commands.

Default — Most programs give you a choice of one or more options. If you do not specify an option, the program automatically assigns one. This automatic option is called the default.

Directory — A special type of object that contains information about the objects beneath it in the naming tree. Basically, it is a file that stores names and links to files.

Disk — A thin, record–shaped plate that stores data on its magnetic surfaces. The system uses *heads* (similar to heads in tape recorders) to read and write data on concentric disk tracks. The disk spins rapidly, and the heads can read or write data on any disk track during one disk revolution.

Diskless Node — A node that has no disk for storage, and therefore uses the disk of another node.

Display Manager (DM) — The program that executes commands that start and stop processes, and commands that open, close, move, or modify windows and pads.

DM Function Keys — Single keys that invoke DM commands.

DOMAIN System — A high–speed communications network connecting two or more nodes. Each node can use the data, programs, and devices of other network nodes. Each node contains main memory, and may have its own disk, or share one with another node.

File — The basic named unit of data stored on disk. A file can contain a memo, manual, program, or picture.

Filter — A command that reads its input, performs a user–specified task, and prints the result as output.

Function Keys — See *DM Function Keys*.

Hard Link — A link that points directly to an object (file).

Home Directory (~) — Your initial working directory. Your user account specifies the name of your home directory.

Initial Working Directory — The working directory of the first user process created after you log in.

Input Window The window that displays a program's prompt and any commands you type.

Insert Mode Insert mode enables you to change text displayed in windows. You can modify text by repositioning the cursor and inserting characters. The rest of the line moves right as you insert additional characters.

Kernel The resident operating system that controls your node's resources and assigns them to active processes.

Link A special type of object that points from one place in the naming tree to another. (Also see *Hard Link* and *Symbolic Link*.)

Logging In Initially signing on to the system so that you can begin to use it. This creates your first user process.

Main Memory The node's primary storage area. It stores the instruction that the node is executing, as well as the data it is manipulating.

Memory Any device that can store information.

Metacharacter See *Shell Metacharacter*.

Name A character string associated with a file, directory, or link. A name can include various alphanumeric characters, but never a slash (/) or null character. You should also remember that certain characters have special meaning to the Shell and must be escaped if they are used.

Naming Directory In addition to its working directory, each process uses a naming directory. Like the working directory, the naming directory points to a certain destination directory. The system uses your home directory as the initial naming directory.

Naming Tree — A hierarchical tree structure that organizes network objects.

Network — Two or more nodes sharing information.

Network Root Directory — The top directory in the network. Each node contains a copy of the network root directory.

Node — A network computer. Each node in the DOMAIN system can use the data, programs, and devices of other network nodes. Each node contains main memory, and may have its own disk, or share one with another node.

Node Entry Directory — A subdirectory of the network root directory. The node entry directory is the top directory on each node. Diskless nodes share the node entry directory of their disked partner node.

Object — Any file, directory, or link in the network.

Operating System — The program that supervises the execution of other programs on your node.

Option — See *Command Option*.

Output Window — The window that displays a process' response to your command.

Pad — A temporary, unnamed file that holds the information you display in a window. A window can display an entire pad, or show only part of the pad.

Parent Directory (..) — The directory one level above your current working directory.

Partner Node — A node that shares its disk with a diskless node.

Password The word you enter next to the "Password:" prompt when you log in. As you type your password, the system displays periods (.) instead of the letters in your password.

Pathname A series of names separated by slashes that describe the path that the operating system must take to get from some starting point in the network to a destination object. Pathnames begin with the starting point's name, and include every directory name between the starting point and the destination object. A pathname ends with the destination object's name.

Pipe A facility that connects the standard output of a command with the standard input of another command.

Print Server A process that oversees the printing of files submitted to the print queue. The print server need only run from the node connected to the print device(s).

Process A computing environment in which you can execute programs.

Prompt A message or symbol that the system displays to let you know that it is ready for your input.

Regular Expression A string specifier that can help you find occurrences of variables, terms, or expressions in your programs and documents. Regular expressions are specified by allowing certain characters special meaning to the Shell.

Root Directory See *Network Root Directory*.

Script A file that you create that contains one or more Shell commands. A script allows you to execute a sequence of commands by entering a single command (the script name).

Shell A command–line interpreter program used to invoke operating system utility programs.

Shell Command An instruction you give the system to execute a utility program.

Shell Metacharacter Any character that has special meaning to a Shell. For example, asterisks, question marks, and ampersands are Shell metacharacters.

Software Programs, such as the Shell and the DM, that allow you to perform various tasks.

Start–up Script A file that sets up the initial operating environment on your node. This file is also known as a "boot script".

Symbolic Link A link that points to link text or the pathname of an object (file). Sometimes also known as a "soft link".

System Administrator The person responsible for system maintenance and security at your site.

Sys5 The version of the DOMAIN/IX system that implements UNIX System V, Release 2, from AT&T Bell Laboratories.

SYSTYPE An environment variable that shows the UNIX system version you are currently using. Valid SYSTYPES for DOMAIN nodes are *sys5* and *bsd4.2*.

Super–User See *System Administrator*.

Transcript Pad A transcript pad contains a record of your interaction with a process. The process output window provides a view of its transcript pad.

User Account The system administrator defines a user account for every person authorized to use the system. Each user account contains the name the computer uses to identify the person (*user ID*), and the person's *password*. User accounts also contain project and organization names. The system uses this information to determine who can use the system, and what resources they can use.

User ID The name the computer uses to identify you. Your system administrator assigns you your user ID. You enter your user ID during the log–in procedure when the system displays the log–in prompt.

Utilities Programs provided with the operating system to perform frequently required tasks, such as printing a file or displaying the contents of a directory.

Wildcards Special characters that you can use to represent one or more pathnames.

Window Openings on the screen through which you view information stored in the system. Display management software lets you create several different windows on the screen. Each window is a separate computing environment in which you can execute programs, edit text, or read text. You can move the windows anywhere on your screen, change their size and shape, and overlap or shuffle them as you might papers on your desk.

Window Legend The area of a window that displays window status information. For example, the window legend of an edit window contains such information as the pathname of the file you're editing, the letter "I" if the window is in insert mode, and the number of the line at the top of the window.

Working Directory The default directory in which a process creates or searches for objects.

Index

Primary page references are listed first. The letter *f* means “and the following page"; the letters *ff* mean “and the following pages”. Symbols are listed at the beginning of the index.

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W