For use with 3.51 Software

AT&T UNIX® PC
Interface Specification

NOTICE

The information in this document is subject to change without notice. AT&T assumes no responsibility for any errors that may appear in this document.

CENTRONICS is a registered trademark of Centronics Data Computer Corp.

GSS-DRIVERS and GSS-TOOLKIT are trademarks of Graphics Software Systems, Inc.

IBM is a trademark of International Business Machines, Inc.

MS-DOS is a registered trademark of Microsoft Corporation

Important Information for Users of the UNIX PC Interface Specification

This update package contains additional guidelines for writing **UNIX** PC applications. Please review this information and keep it with your Interface Specification.

1 Page 1-2. For remote users, a full function keyboard is now available with the AT&T Personal Terminal 610.

2 Page 1-5. The **Printers** Office object is used to configure a printer, restart it, and to view and delete files from the printer queue (**Printers**).

3 Page 1-65. Two new Preference settings are available with Version 3.5. **Mult-user Items** determines whether **Other Users** and other features that support a multiuser environment are displayed in the Office. **Turn screen off after _____ minutes** provides automatic screen dimming. It can be set for 1-59 minutes after the last keystroke or mouse movement. With the default value of 0, the screen never dims.

4 Page 1-67. A full borderless window can be created as follows:

Under the login directory, /u/userid/, create a file named Office with the following contents:

Name = **Borderless UNIX window** (or any other choice)

Default = **Run**

Run = **EXEC -d /bin/sh** (or pathname for other shell)

After logging out and logging back in, **Borderless UNIX window** appears in the Office of that user only. More than one borderless window can be opened; each will be listed in

the Window Manager with the label **unknown contents**. See ua(4).

5 Page 2-14. Version 3.5's printer subsystem configuration supports one additional remote printer and six additional serial printers.

6 Page 2-47. The new C compiler supports flexnames. Programs that have symbol names longer than eight characters are not truncated automatically as was done with the pre-flexname compiler.

7 Page 2-48. A new option, -T, is provided to truncate long symbol names to eight characters. This is useful so that new object files can be linked with older (pre-flex name) files and then run under either old or new systems.

8 Page 2-48. The Link Editor, ld, provides a new option, -G, that resolves symbol referencing errors between new flexnames, generated files containing long symbol names, and the older (pre-flexnames) object files. Under Version 3.5, this option assures backwards compatibility with old files.

9 Page 2-61, 2-66. When removing a line from a file, you should look for an exact match. For example, if you grep just for **EDIT**, other application-created variables, such as **MAILEDIT**, would be removed as well.

10 Page 2-63. To abort an installation, **Install** must exit with a return code of 64. This suppresses addition of entries.

CONTENTS

*UNIX** PC USER INTERFACE SPECIFICATION

PREFACE

The *AT&T* **UNIX*** *PC Interface Specification* is intended for software vendors and developers as a guideline for writing **UNIX** PC applications. This document contains specific **UNIX** PC programming information. It supplements the general **UNIX** System V information in the *AT&T* **UNIX*** *PC UNIX System V Programmer's Guide* and the *AT&T* **UNIX*** *PC* **UNIX** *System V User's Manual* with information that is essential for applications to be installed and used with the **UNIX** PC's graphics, keyboard, mouse, printer system, windows, menus, and remote terminals. References in the form of a **UNIX** System V command name followed by a number in parentheses, such as tam(3T), can be found in the *AT&T* **UNIX** *System V User's Manual.*

Information in this document is organized into two sections:

- ***UNIX*** **PC USER INTERFACE**— contains a user-level description of the software and definitions for all aspects of the interface. It provides the framework for an applications designer, developer, or programmer to understand the user interface of the **UNIX** PC.
- **DETAILED INTERFACE SPECIFICATION**— contains descriptions of hardware and software features, the Terminal Access Method (**TAM**) interface support routines, and the software installation/removal procedures. It provides specific guidelines for an applications programmer to write software that uses **UNIX** PC capabilities.

---

* Trademark of AT&T

Section 1

UNIX PC INTERFACE SPECIFICATION

PAGE

Section 1

UNIX PC INTERFACE SPECIFICATION

Chapter 1

INTRODUCTION

Section 1 describes the **UNIX** PC user interface, which presents the multitasking power of the **UNIX** System V operating system in a form usable in the mass marketplace. The primary interface is the User Agent, which provides users with a window- and menu-based access to the **UNIX** System V file system, to applications, and to communications.

The tools used to build the User Agent are available for use by any application. Applications developers for the **UNIX** PC are encouraged to use these tools because consistency is a major component of "user friendliness" in any system.

The user interface design guidelines used for the development of the User Agent are based on the way a person works. We believe that work is usually approached by getting what is to be worked on, an "object," and then applying tools to the object to do a task. For example, when someone wants to edit a document, he or she first gets the document and then finds the red pencil to mark it up.

We designed the User Agent to allow the user to locate the object to be worked on and open it up; the User Agent "finds" the tools needed for the worker to do the job; for example, by automatically invoking the application used to create the object.

Rarely in an office can a worker complete work without interruption. The **UNIX** PC lets the user have several tasks, in various stages of completion, displayed on the screen at once. A user does not have to close a task at an inconvenient point to handle an interruption; the task can be set aside and resumed when convenient. Switching between tasks and managing displayed tasks are easily accomplished using the window management facilities of the **UNIX** PC.

As users begin relying on the **UNIX** PC for their office work, they might want to access it from a remote location to retrieve data, read electronic mail, and perform other tasks. We developed methods of displaying the data and entering commands that are compatible with remote access. Remote users do not have to learn an entirely new way of interacting with the **UNIX** PC even though many of the features that make the **UNIX** PC easy to work with—a mouse, a bit-mapped display, simultaneous running of multiple applications, and the full-function keyboard—are not available now from remote terminals.

Remote users, like all users, also can interact with the **UNIX** PC by using the System V shell if they choose.

Basic Design Concepts

The basic concepts underlying the User Agent follow.

Office Metaphor

The User Agent uses familiar office terms instead of jargon when office metaphors are reasonable.

The first screen the user sees after logging in is shown in Figure 1-1. It contains a single window called the *Office* that lists the names of various office objects. The user can access an object by opening it as described in **Point and Act Method** in Chapter 3—**INPUT**.

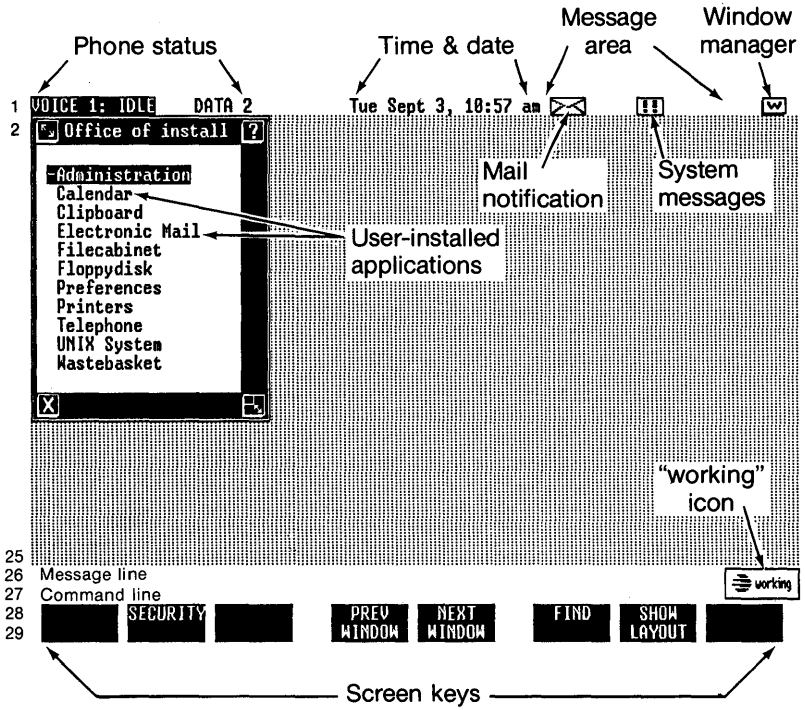


Figure 1-1. Screen Layout

The Office software includes easy-to-use tools for managing the file system, which has several elements (as presented to the user).

- The *Filecabinet* contains all user-created files that are stored on the hard disk. With it comes a friendly access to the **UNIX** System V directory hierarchy. The Filecabinet contains File folders and Files.

  — A *File folder* is an interface to a **UNIX** System V directory.
  — A *File* is the basic unit of work, labeled according to the application which created it, and is a **UNIX** System V file.

- The *Floppydisk* allows users to access user-created files that are stored on floppy disks. *Floppydisk* contains a menu that allows the user to display the contents of a mountable file system on a floppy; format, copy and repair floppies containing **UNIX** System V file systems; and format, read, and write floppies containing **MS-DOS*** file systems.

- The *Wastebasket* holds Files and Folders that have been deleted from the Filecabinet, a Folder, or the Floppydisk. Users can retrieve these items from the Wastebasket. The *Wastebasket* can be set to empty automatically, at user-specified intervals. This is done in the *Office Preference* menu.

- The *Clipboard* can be used, where provided for in the applications software, to transfer data between applications. The *Clipboard* file format is described in adf(4).

---

* Trademark of Microsoft Corporation

Users access the contents of the Filecabinet, Floppydisk, Wastebasket, Folders, and Files by opening them. A new window is created and the contents of the object are listed in menu form. When the user opens a File, the application needed for the user to edit or manipulate the File is automatically loaded. The user does not have to load the application before selecting the file. File type specification is done via a suffix on the File name that is not normally displayed to the user. Suffixes are defined on installation of software in the file **usr/lib/ua/Suffixes**. See ua(4).

Voice and data communications are accessed from the *Telephone*. Other object functions accessible from the Office include:

- Setting user preferences (*Preferences*).
- Performing system administration and maintenance (*Administration*).
- Viewing and deleting files from the printer queue (*Printers*).
- Opening a window on the Bourne shell (**UNIX** System, which is shown in the Office only when the user is set up in Expert mode by the "install" login).

Other items or utilities can appear in the Office, depending on the applications software that the user installs. In Figure 1-1, Electronic Mail and Calendar are two objects not included in the Office software that might have been installed separately by the user. See Chapter 5—**SOFTWARE INSTALLATION** in Section 2 for a description of how an application install script places entries in the Office and in other menus.

Task Segregation By Windows

The user can view different parts of the file system, including files for different applications, simultaneously in different windows. The user can have several windows open on different applications, and switch between applications by changing windows.

The user changes the status of windows (opens, suspends, resumes, and closes them) through the window management facilities of **TAM** and the Window Manager. Only one window is active at a time; the others are suspended. Opening a new or suspended window suspends the presently active window. However, any action initiated by the user in a window continues until the next user input is required. The window management facilities of **TAM** and the Window Manager are described in tam(3) and window(7).

User-Modifiable Windows

Users can change the size, shape, and location of some windows to suit their needs. Windows can be modified using the mouse, by clicking on the move and shape icons, or using the keyboard via the Window Manager Commands menu. **TAM** does not support resizing for all window types, particularly forms. If a window cannot be resized, resize icons should not be enabled. See wcreate() in tam(3) and window(7).

Screen Regions

Information is presented consistently on the screen in a standard location according to type (see **Screen Layout** in Chapter 2—**DISPLAY**). Status information, command echo, and prompts and feedback each have standard locations. The command entry, prompt/feedback, and screen labeled key regions are owned by the application running in the active window.

Core Commands

The same commands are used to perform analogous actions on different types of objects across different contexts. These core commands are on fixed (hard-labeled) function keys on the keyboard. Context-specific commands are on screen-labeled keys or in menus. See **Keyboard Commands** in Chapter 4—**DEFINITIONS OF KEYS AND ACTIONS**.

Command Construction Using Menus And Forms

The primary means by which the user instructs the system is through menus and forms. Thus, users do not need to learn a complex list of command names and options or a complex command syntax. Instructions for users to find out what they can or should do next are readily available on menus or forms. Commands are constructed by combining names of actions and objects where appropriate. See Chapter 3—**INPUT**.

Choice Of Methods Of Command Entry

Three general methods are available for users to specify actions and objects: select an item from a menu, press the appropriate function key, or type the name. Menus inform users about available actions and objects, and so serve as a form of tutorial. As a user gains experience, he or she may find bypassing menus convenient. The multiple methods of command entry give all users a choice and can be of particular benefit to remote terminal users. Typed equivalents of many commands are echoed on the screen, regardless of the method of command entry, to provide the user with feedback.

Use Of Defaults

To minimize the number of decisions and specifications users must make, a default action and a set of attributes are defined for each object. Default options are defined for actions, when options are available, and these defaults are shown in options forms.

Default actions are specified for each object type in **usr/lib/ua/Suffixes**.

Protection For Naive Users

Two forms of protection are available at the user's option: access to the **UNIX** System V shell can be prevented and functions that potentially can cause problems, for example, result in loss of data, require confirmation. On line help is provided from any point. When requested, On line help pertains to the current environment. Users may also examine any screen in the Help system.

Layered Complexity

Only a few, simple actions are necessary to use the User Agent. Common actions are easily performed, most with a single keystroke. More complex actions are available, and these are built by combining simple actions. Shortcuts to combine some simple actions are provided for the more experienced user.

Remote Access Compatibility

Users can access the Office and many applications from a remote, character-oriented terminal with little change in available functionality. Functions that require a mouse or bit-mapped graphics cannot, of course, be accessed. Only one application can be present on the remote terminal screen at a time. Status line functions are not available remotely. The user interface on the remote terminal is compatible with the user interface on the **UNIX** PC.

Detailed descriptions of the functions and capabilities of the **UNIX** PC are presented in Chapter 2—**DISPLAY**.

Chapter 2

DISPLAY

This is a description of the layout of the display and its parts, including windows, status areas, and command entry areas.

Screen Layout

The **UNIX** PC screen is an area 720 pixels wide and 348 pixels high, corresponding to an 80-column by 29-line character display. Characters in the system font are defined within a 9 by 12 pixel matrix—a row is 12 pixels high and a column is 9 pixels wide.

An example of the **UNIX** PC screen is shown in Figure 1-1. It consists of 29 lines divided into the five areas described below.

Status Line: Line 1

The Status line displays telephone status information and icons signalling the arrival of messages. An icon allows the user to access a list of active windows. The line is subdivided into functionally different areas. The user can point with the mouse to an area or an icon to activate these different functions. (Keyboard users can access these functions from equivalent keys that are also described.) These areas are:

- Telephone line status for up to two lines and voice/data status (columns 1 through 32): Pointing anywhere in this area with the mouse and pressing <B1> displays the Call Screen. (Keyboard access: <Shift>-<to F2>)

- Date and time (always displayed)

- Message arrival: An application can turn on an icon in this area and sound the beep to signal arrival of an asynchronous message. The user accesses the message by pointing to the icon with the mouse and pressing <B1>. See **Asynchronous Messages** in Chapter 12—**FEEDBACK, PROMPTS, AND MESSAGES**. (Keyboard access: <Msg>)

- Window Manager: Pointing at this icon and pressing <B1> displays a list of currently open windows. (Keyboard access: <Suspd>)

Applications can write to the Message Arrival area only through the Status Manager. See **Asynchronous Messages** in Chapter 12—**FEEDBACK, PROMPTS, AND MESSAGES** for a description of how the user accesses the functions contained on this line.

Work area: Lines 2-25

Contains the work area, including all user-controlled windows.

Messages/Prompts: Line 26

Contains synchronous feedback from user-issued commands and contains prompts. A maximum of 70 characters can be displayed. See **Prompts** in Chapter 12—**FEEDBACK, PROMPTS, AND MESSAGES**.

Command Entry/Echo: Line 27

Commands are displayed as they are typed or entered via function keys and menus. A maximum of 70 characters can be displayed. See **Command Echo** in Chapter 12—**FEEDBACK, PROMPTS, AND MESSAGES**.

System Busy: Lines 26, 27

A "working" icon is displayed on the right. See Figure 1-1.

Screen Keys: Lines 28-29

Contains the legends for the eight Function Keys (<F1> through <F8>). The pads for the labels are always shown as 2 by 8-character inverse video regions, whether or not any labels are present in a particular pad. All sixteen characters in each pad can be used for a legend. The pads are grouped in a 3-2-3 arrangement, to match the grouping of keys <F1> through <F8> on the keyboard. There is one dark space between pads in each group and five dark spaces between groups. Screen keys contain commands that refer to the current active window or current popup window. Users activate the function behind the screen key by pressing the corresponding function key <F1> through <F8>, or by pointing with the mouse to the screen key and pressing <B1>. Labels for screen keys are changed via the tam(3) call **wslk()**.

Recommendations for Screen Keys

- Use mixed upper and lowercase characters on screen keys to improve readability.
- Center both top and bottom lines.
- If a legend takes only one line, use the top line.
- Functions that appear in several contexts or windows should appear on the same screen key, when possible.
- Display only functions that are available in the current context or environment.
- Display all eight screen key labels. Screen keys assigned a function by an application should display a label. Screen keys not assigned a function should be blank.

Note: Shifted <F1> through <F8> are reserved for use by the **UNIX** PC Telephone Manager. They are not available for use by applications.

Windows

A user views each task through a window. A window has one or more lines of data entry/display area surrounded on all four sides by a border. A borderless window can also be defined, for applications that require the use of the full screen, 24 rows by 80 columns.* At the other size extreme, a window can be just large enough to display scrolling controls, if present, with at least one blank row (12 pixels) inside the border. See **Window Appearance—Application Windows** in this chapter for definition of window creation and size definition.

The default window size is set by each application. Some applications require a full 24 by 80 display (for example, 3270 emulation and asynchronous terminal emulation) and the borders are not shown.

Each application is run in a window. Menus of commands, menus of object names, and forms are all displayed within windows. Several windows can be open simultaneously on the display. Only one window at a time, the *active window*, receives input from the keyboard and has information shown on the command entry line and synchronous message line. The labels on the screen keys refer to the active window, and it is "on top"—fully displayed. It can cover all or parts of other windows on the display.

---

* An application can define a window to cover the status line, e.g., as 25 rows by 80 columns. However, a better procedure is to load a font that is 11 pixels high in Slot 0; this allows writing 26 lines of characters in the work area.

The border of the active window is highlighted, with dark text and dark icons. The borders of inactive windows are dark, with bright text and icons.

When a window is opened, its initial position is determined by a location algorithm designed to minimize overlap among windows. Users can change the shape and position of most open windows as they desire.

Windows are not explicit Office objects; they do not exist independent of their contents. A window is a special object that is synonymous with an open object. A window is created whenever a user issues a command to "open" an object. A window is removed when the user exits from the application, menu, or form, or cancels the command that generated the window.

Window Border Areas

The top line of a window contains an identifier for the window contents, where the form of the identifier depends on the window contents. The window border contains icons that allow the user to access frequently used window control functions and to access "help" with the mouse. The icons are shown in Figure 1-2. To access the functions, the user points to the appropriate icon with the mouse and presses <B1>. (For the arrow icons, <B2> and <B3> also activate functions.) Each of the functions listed below maps onto a function available from the keyboard. See Chapter 4—**DEFINITIONS OF KEYS AND ACTIONS** for definitions of the commands. See tam(3) for a description of how to create a window and turn on border items. Note that the application must handle the following codes returned by selecting one of the icons:

[?] Pressing <B1> provides context-sensitive help. Same as <Help>.

[X] Pressing <B1> (cancel/exit) cancels a selection or a partially specified command. If nothing is selected, exits from the application and closes window. Functions as <Cancl> or <Exit>, depending on context; maps onto <Shift>-<Cancl> key.

arrows Pressing <B1> (scroll text up, down, left or right) reveals text in the direction the arrow points. Maps onto <Shift>-<arrow key>.

Pressing <B2> (page text up or down one "windowful") reveals text in the direction the arrow points. Maps onto <Page> (down), <Shift>-<Page> (up).

Pressing <B3> (move all the way up or down) moves to a beginning or end. Maps onto <Beg> (up), <End> (down).

The following border icons are used to control the window. These functions are handled in **TAM** window management.

[move] Pressing <B1> (in the upper left corner) moves the active window without changing its shape. The user

1) presses <B1>,

2) waits until a dithered outline appears,

3) drags the outline to the next position, and

4) releases <B1>.

Then the window moves immediately to the new position to match the outline. The shape of the window does not change, and the user cannot move any portion of the window off the display.

[shape] Pressing <B1> (in the lower right corner) shapes the active window without changing the position of the upper left corner. The user

1) presses <B1>,

2) waits until a dithered outline appears,

3) drags the outline to the new shape, and

4) releases <B1>.

Then the window changes its shape immediately to match the outline. The upper left corner of the window remains anchored.

Conventions

- When an icon is not available, it is not displayed. For example, if window contents cannot be scrolled horizontally, the left and right arrows are not displayed. If no help is available for the window, the help icon is not displayed. See tam(3) wcreate.

- The title of the window is in mixed upper and lower case letters and reflects the name of the menu item or command used to open the window. See tam(3) wlabel.

Window Appearance—Application Windows

An application window lets the user view an application program or process. The format of the internal display depends on the application.

Applications are not required to know the size of the window in which they appear, although applications can be written to make use of the Window Manager routines to react intelligently

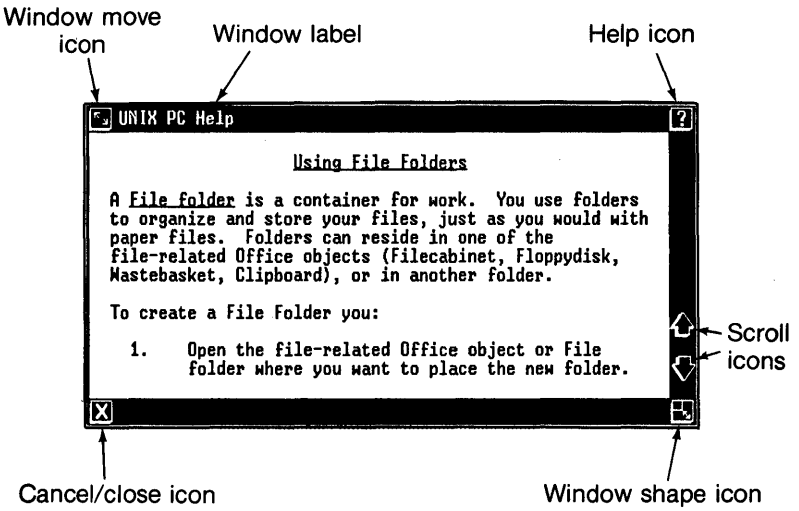


Figure 1-2. Window Icons

to changes in window size and shape. Applications that do not interface with Window Manager routines still run in windows, but do not react to window changes. Window wrap and truncation is consistent with the ANSI X3.64 standard.

Application Window Border Areas

The top line of a window on in integrated application (one designed or modified to run with the Window Manager) contains an identifier for the application that is currently running in the window and the name of the object that is currently open in the window. If the application owning the window allows switching between "overtype" and "insert" text

editing modes, the indicators "OT" and "IN", respectively, should be displayed to the right of the window identification information. Border icons function as previously described in **Window Border Areas** in this chapter.

Window Appearance—Menu Windows

A menu window is a window listing the contents of a menu or a **UNIX** System V directory. Office Filecabinet, Floppydisk, folders, and Wastebasket windows are examples. The content of a file system window is displayed as a menu, where the first word in each line is the name of a folder or file and the remainder consists of a description. (The description is displayed at the user's option, as set in ,Office Preferences.) The description is the name provided in the Suffixes file. When both name and description are displayed, they are left-justified, separated by a dash. Uppercase names appear before lowercase names. See Figure 1-3.

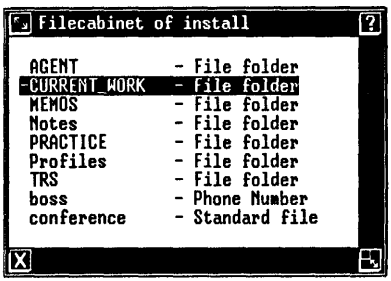


Figure 1-3. Filecabinet Menu

Menu Window Border Areas

The label area of a file system window contains the name of the open object and the name of the folder that contains the file, if it is in a folder; that is, the last two elements of the path name are displayed. For example, Filecabinet/practice is the window label for a folder named "practice" which is stored in the Filecabinet. See ua(4). Other icons are as previously described in **Window Border Areas** in this chapter. The label of a menu window contains the string defined when the menu is called. See menu(3T).

Default Window Properties

Windows on menus are sized according to their contents instead of being a standard size. A window is no larger than needed to display its contents, up to a default maximum size (that can be less than maximum window size). When the number of items is more than can be shown in the maximum size, items that are not displayed can be viewed by scrolling or paging.

Menu windows must be wide enough to display one column of their contents. They cannot be resized to a narrower format.

Menus may exist until the user explicitly closes them (e.g., Office Administration, Filecabinet) or they may disappear as soon as the user makes a selection (e.g., Commands, Create). See menu(3T).

Mouse Pointer

The mouse pointer is a symbol that follows the motion of the mouse and moves anywhere on the display. In the User Agent the symbol is an arrow. The mouse pointer location is independent of the active window or selected items. Items are selected and windows activated only when the user presses a mouse button or issues a command.

The tip of the arrow is the "hot zone." (The "hot zone" of other selection pointers should be intuitive, for example, the center of an "X" or "+".)

More than one mouse pointer can be defined to signal different states. The mouse pointer can be changed within application-defined rectangles on the display.

Working Icon

A "working" icon, under kernel control, is displayed on the right end of lines 26 and 27 as a "system busy" indicator. See Figure 1-1.

Status Line Icons

Four icons signal arrival of asynchronous messages: an "envelope" that signals arrival of mail, a "calendar" that signals an appointment (optional software), a [!!] icon to signal arrival of a message from the system, and a [!] for other messages. See Figure 1-1. The electronic mail envelope icon is turned on when mail arrives in **/usr/mail/userid**, regardless of the mail application installed. Other icons can be defined for other applications.

Window Border Icons

Icons used in window borders are as previously described in **Window Border Areas** in this chapter.

Chapter 3

INPUT

This Chapter describes the syntax and methods used to enter commands and display file system windows in the Office. More than one method of command entry is provided to match the diverse needs of remote users, users with different skills and preferences, and situations that make one method more efficient to use than another.

In most cases, commands are formed by combining the name of an object with the name of an action. Actions can be specified by pressing a function key on the keyboard, typing the name of the action, or pointing at and selecting the name of the action in a menu of objects. Objects can be specified by typing the name of the object or pointing at and selecting the name of the object in a menu. More than one object can be selected for each action. The user controls the selection (highlighting) using the keyboard or the mouse. A mixture of keyboard and mouse modes of command entry is possible, even within a single command.

For each object type shown in the Filecabinet or Folder, a default action is defined that is executed if no other action is specified. The default action and specific action-object pairing are defined in the **/usr/lib/ua/Suffixes** file. For each action that takes an object, the default object is the object under the highlight or cursor, even if the object has not been explicitly selected.

Command Entry: Keyboard And Mouse

Following is a description of the devices by which the user can enter commands and the available methods. Two input devices are available, the **UNIX** PC 103-key keyboard and the **UNIX** PC three-button mouse. All commands can be entered with either device, but it is not necessary to have a mouse to perform any User Agent Functions. This ability to essentially perform all User Agent functions from the keyboard allows remote access to the User Agent.

UNIX PC Keyboard

The **UNIX** PC keyboard is shown in Figure 1-4. The keyboard is divided into five functional regions:

1. Standard typewriter, QWERTY, area

2. Action keys (left cluster)—hard function keys

3. Function keys (top row)—keys <F1> through <F8>, whose functions are defined by the application controlling the active window; these functions are shown on the eight screen keys.

4. Process control keys (nine keys on the top right)—hard function keys.

5. Display control/numeric pad (twelve keys on the lower right).

Generic definitions of all keys on the keyboard are provided in Chapter 4—**DEFINITIONS OF KEYS AND ACTIONS**. Applications should use these keys in a manner that is semantically consistent with these definitions.

Each state of each key on the keyboard is mapped onto a mnemonic escape sequence. For example, typing <Esc> <c> <m> is equivalent to pressing the <Cmd> key. This provides

remote users with access to the keyboard keys, particularly those that do not appear in menus. All of these escape sequences may be found in **/usr/lib/ua/keymap**. Application developers should not use the <Esc> key within the application since these escape sequences are recognized from the console and remote terminals.

It is possible to customize terminals for use with the **UNIX** PC by defining function keys on the remote keyboard as equivalent to the **UNIX** PC keyboard. This is done by creating a terminal.kmap file in **/usr/lib/ua** and inserting a pointer to that file in the termcap entry for that terminal. See abd(7) and termcap(5). Also see **/usr/lib/ua keynames**.

UNIX PC Mouse

The **UNIX** PC mouse controls the position of a pointer and selects items (actions, objects, icons, and fields). The mouse has three buttons with generic definitions equivalent to three keyboard keys. The definitions can be modified if the mouse is pointing at an icon.

The buttons are defined as follows:

<B1> The left mouse button, equivalent to <Enter>, is the "do it" button. It selects and initiates action on the item pointed to by the mouse pointer.

- If the item is an object, selection executes the default command on that object.
- If the item is an action, selection executes the action.
- If the item is an icon, selection executes the action or selection represented by the icon.
- If the item is a portion of an inactive window, selection activates that window.

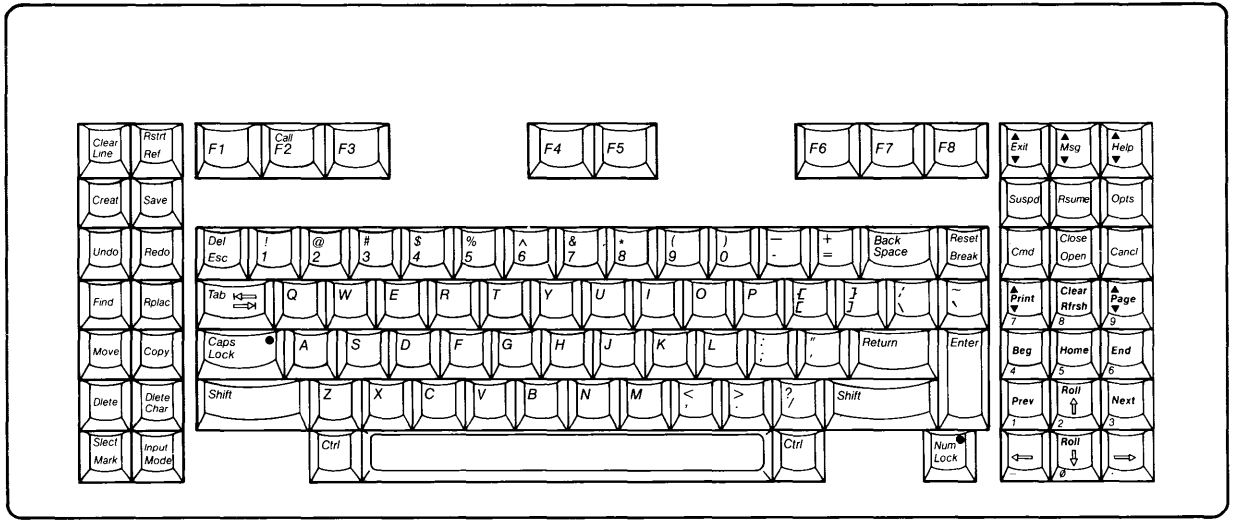


Figure 1-4. UNIX PC Keyboard

- If the item is the dotted "work area" (outside of any window), selection suspends the current window and activates the window manager.

- For window border arrow icons, selection scrolls the contents of the window in the direction of the arrow.

- If the item is [OK] in a form, selection executes the form.

- If the item is a message, pointing anywhere inside the window and pressing <B1> closes the window and is equivalent to pressing <Enter> [see message(3T)].

<B2> The middle mouse button, equivalent to <Cmd>, is the "show me what I can do" button.

- It displays a menu of commands that are currently available for the object under the highlight or current context.

- It displays a menu of entries that are allowed in a constrained field in a form.

- For window border arrow icons, it pages window contents in the direction of the arrow.

<B3> The right mouse button, is usually equivalent to <Mark>. It allows the user to mark more than one item as an object for subsequent action.

- In a menu, it selects the unselected item pointed at by the mouse pointer.

- In a menu, it unselects the selected item pointed at by the mouse pointer.

- It steps through the allowed entries in a constrained field on a form.

- For window border arrow icons, it displays the top or bottom of the object in the window.

Command Entry Methods

Selection by Pointing

Objects and actions can be selected from menus by pointing and selecting. There are four ways to point to an item.

1. Move the mouse pointer to the item in a menu, so that it becomes highlighted.

2. Move the highlight to the item in a menu using the arrow keys.

3. Move the highlight to the item in a menu by typing the first few characters of the name of the item. Only as many characters as are needed to uniquely identify the item in the current menu need to be typed, but the user can type the entire name of the item. Capital and lowercase characters are treated equivalently.

4. Move the highlight to the next item by pressing <Return>. This works **only** if no characters have been typed.

The user can point to the name of an object in an inactive window by typing the path name of the object. For example, the user could open the sample file, "edit," by typing **Filecabinet/practice/edit** and pressing <Return> from the Office or any file system window.

> ***Note:*** Filecabinet is an alias for **/u/userid/Filecabinet** (see **Typed Command Line Entry** later in this chapter).

Point and Act Method

With the point-and-act method, the user points to the item (object name, action name, or icon) and then presses a function key or mouse button. The item is automatically selected and the specified or default action on that item is initiated.

The simplest command, select and open, is issued by:

1. Pointing at the object name, using
    - the keyboard to type the first letter(s) of the name,
    - the keyboard cursor movement keys, or
    - the mouse to position the pointer, and
2. Selecting the name, by
    - pressing the <Enter> key on the keyboard, or
    - pressing the left button, <B1>, on the mouse.

This opens the object in a window and gives the user access to its capabilities. Open is the default action on almost all objects. The default action is defined for each object type in **/usr/lib/ua/Suffixes**. Other commands can be issued from the Commands menu or via function keys or hard keys.

To issue a command using the Commands menu, the user points to the object and then presses <Cmd> on the keyboard or <B2> on the mouse. This selects the item implicitly and displays a menu of available commands. Then the user points in the Commands menu to the desired action and presses <Enter> or <B1> to initiate the action. To specify an action using the hard keys or function keys on the keyboard, the user points to the object and presses the desired key.

At the next level of complexity, the user can select several objects for a single action. The user points at each object and presses <Mark> on the keyboard or <B3> on the mouse. The user can also select a group of contiguous objects by 1) pointing with the mouse pointer to the first, 2) holding down <B3>, 3) dragging the mouse pointer through the adjacent items, and 4) releasing <B3>. Then the user presses an action key or <Cmd> to select an action for all selected objects.

Typed Command Line Entry

Typed command entry using the command language of the User Agent (as opposed to **UNIX** System V shell commands) is needed to support the expert User Agent user, who can become very adept in using the User Agent while knowing nothing of **UNIX** System V commands. Users might not use the typed method much at first, but as they become familiar with the system and learn the names of the commands, many users choose to type because it can be a faster method of interacting with the system. Typed command entry is also important for supporting remote users, who may find working with menus slow and cumbersome. The User Agent provides a method for turning off command menus; other applications could follow.

Typed Command Syntax A simple command syntax is used for typed commands.

```
<Cmd>                    (key)
action name              (one or more characters required)
<SPACE>                  (blank)
object name(s)           (typed or previously selected)
<Enter> or <Return>  (key)
       |
       |
       V
Action taken
```

The syntax is extended to allow for "operator first" entry when typing the entire command, as well as "operand first" entry to

allow for selection of an object from a menu). The syntax for "operator first" is:

```
<Cmd>
action name
<SPACE>
object name(s)
<Enter> or <Return>
        |
        |
        V
 Action taken
```

When the <SPACE> is typed, any objects that were selected before <Cmd> was pressed are unselected. The syntax for "operand first" entry is:

```
select object(s) from menu
<Cmd>
action name
<Enter>-<Return>
        |
        |
        V
 Action taken
```

1. All typed User Agent commands are preceded by pressing the <Cmd> key to display a menu of available commands, clear the command echo line and enable input on this line. (Through the Office window's Preferences, users are able to turn off display of the command menu if the application supports it.)

2. Tokens (words or their abbreviated representations) are separated by spaces.

3. When <Cmd> is pressed, one action name must be specified for each command—an object name alone is not accepted. (The default action assumed if <Cmd> is pressed is the first one in the Commands menu. Hence, no ambiguity exists about whether a token is an action or an object.)

4. For multipart commands (for example, Move and Copy) the user enters the terminator after listing the first set of arguments. Then the User Agent prompts the user to enter further information as needed.

5. Any valid action name can be typed, including actions on hard and soft function keys, whether or not the command menu is currently displayed.

Specifying Actions in Typed Commands Action names can be truncated; the user only needs as many letters of the action name as needed to identify it uniquely.

Specifying Objects in Typed Commands

1. An object name can be typed whether or not its name is currently displayed in the active window.

   An object can be specified in one of the following ways:

   - If an object name is shown in the current active window, it may be specified by typing the full path name.

     As with actions, object names can be truncated to the shortest string that uniquely specifies the object. The wildcard character "*" can be substituted for one or more characters, and it allows specification of one or more objects.

- If an object name is not shown in the current active window, supply an abbreviated term for certain path names. Six special aliases for path names are recognized:

System	root (/)
Filecabinet	/u/userid/Filecabinet
Floppydisk	/mnt
Wastebasket	/u/userid/Wastebasket
Clipboard	/u/userid/Clipboard
Parent	the object that contains the current folder—equivalent to **UNIX** System V "..".

Following are some examples.

— If the user wants to delete "filea," which is stored in "folderb" in the Filecabinet, and Filecabinet is not open, typing:

<Cmd> delete Filecabinet/folderb/filea <RETURN>

places filea in the Wastebasket.

— If the user wants to execute the default command on filec (open it), but the Filecabinet is not open, typing:

Filecabinet/folderb/filec <RETURN>

selects the object and executes the default command, Open, on it. The user does not have to type the full **UNIX** System V path name.

Applications must also recognize these abbreviated path names and should not block user's attempts to use them to identify user-created folders or files.

- Type the full path name of the object from root, for example:

/u/tutor/Filecabinet/foldername/filename

2. A typed object name overrides any object(s) that are currently selected, allowing the user to take actions on objects that are not in the current menu.

Editing <Back Space>, <Clear Line>, and <Dlete Char> can be used to edit uncompleted commands. <Cancl> clears the command line of any partially entered items and returns the cursor/marker to its position prior to beginning the command.

Regular Expressions **UNIX** System V regular expressions, including **?** [ ] **!**, are not recognized in typed User Agent commands. The wildcard character ***** is recognized and interpreted.

Specifying Options

- The Security command sets access and execution permission on the selected object.

If options are available on a command, they are displayed in a menu or a form when the command is invoked by menu, function key, or typing the command name. The user selects the options from the menu or fills in the form. Menus and forms have defaults, so that the user only needs to press <Enter> to continue.

Command Echo

The user must be given feedback. When commands are issued on function keys or selected from menus, an opportunity exists for erroneous selection. The user must be given feedback as to what was done to have the opportunity to take corrective action.

- All object-control commands [action name and object name(s)] are echoed, whether selected from a command menu, screen key, hard function key, or typed. If objects are selected from a menu (not typed), the names of the objects are not echoed until the action is specified. This is true whether the action is selected from a screen menu or from the keyboard.

- The syntax of the command echo is the typed command syntax, for example, action-object, as just described.

- For default actions, echo the name of the action as if it were specified (for example, "open objectname").

Type-Ahead and Autorepeat

Type-ahead is supported within windows. However, if the user initiates creation of a new window and begins typing immediately, a few keystrokes can be lost.

Certain keys, such as space bar and backspace, autorepeat. See kbd(7). Mouse buttons autorepeat when they are held down. Keystrokes that should be generated by autorepeat are flushed as soon as the key or button is lifted, so that the repeated function does not continue, and the user has more control over the display.

Chapter 4

DEFINITIONS OF KEYS AND ACTIONS

This is a description of the generic definitions of core commands, both those on hard keys and others available through the User Agent.

Core actions are actions used to perform analogous functions for different types of objects across different contexts. The User Agent selects the appropriate function for the specified action. The same action name can be used for functionally similar actions in different contexts and on different object types.

The high level semantics (meaning) of each command should remain constant, but responsive to context. For example, the Copy command always involves duplicating data from one location to another, without deletion of the original data. (It should not be used, for example, to specify the number of copies of a document to be printed.) The same command name should be applied to a file, a folder or a piece of a graph. A different command name should not be invented for each different type of object.

Keyboard Commands

The keyboard on the **UNIX** PC is the AT&T Product Family keyboard. The command definitions that follow are consistent with AT&T product family usage.

The commands listed, in alphabetical order, are found on the keyboard. Not all of the commands are used in the User Agent.

Further discussion about using process control commands (Enter, Cancl, etc.) is in **Process Control Commands: Conventions** later in this chapter.

<Arrows> Each keystroke moves the highlight or cursor one unit in the indicated direction. When a border is reached, a keystroke moves text under the cursor or highlight one unit to reveal text in the direction of the arrow (scroll); the cursor or highlight remains stationary on the display. "Unit" is defined by the application in the window, for example, one or more lines, one or more characters, or columns, etc. Both horizontal and vertical cursor or highlight movement can be supported. It is an autorepeating key.

Each shifted keystroke (scroll) moves the text under the cursor or highlight one unit to reveal hidden text in the direction of the arrow; the cursor or highlight remains stationary on the display. It is an autorepeating key.

<Back Space> Deletes the character to the *left* of the cursor and closes the gap. The cursor moves one character position to the left. It is an autorepeating key.

<Beg> Moves the cursor or highlight to the first character or field in the current window. <Shift>-<Beg> moves the cursor or highlight to the first character or field in the current object (top).

<Call> This <Shift>-<F2> combination invokes the Call screen from any context or application.

It is a dedicated key and can only be used by the **UNIX** PC Telephone Manager.

<Cancl> Cancels any partially specified command and unselects any selected objects. If given from a form, it cancels any changes to the form window, cancels the command that displayed the form, and closes the window. See **Process Control Commands: Conventions** later in this chapter.

<Caps Lock> Modifies the meaning of alphabetic keys. When the light is on, alphabetic keystrokes are echoed as capital letters. Other keys are not affected.

<Clear> This <Shifted>-<Rfrsh> combination clears a region as determined by the application. It is not used by the User Agent.

<Clear Line> Clears the current line or field in a form; it clears the command line.

<Close> This <Shift>-<Open> combination closes the current window. It is synonymous with Exit. See **Process Control Commands: Conventions** later in this chapter.

<Cmd> Opens the command line and displays a menu of all commands available in the current context. The menu can be turned off at the option of the user via Preferences. It displays a menu of options in a constrained field in a form and maps onto <B2> on the mouse.

<Copy> Copies the selected object(s) or area(s) from one location to another without deleting the original. After the area is specified and the Copy command is given, the user is

prompted to point to the destination location and give the PASTE command. See *Other Commands* following this section. After the user gives the PASTE command, the destination window is updated immediately. Copy cannot be used to overwrite an existing file, like the **UNIX** System V shell "cp." If a file with the same name already exists in the target location, the user is prompted to give a new name to the copy. When applied to a Folder (directory), the command copies both the Folder and its contents.

<Creat> Allows the user to create an object. In the User Agent, it displays a menu that allows the user to select an object type, then it displays a form that prompts the user to assign a name to the new object, and finally it opens the object in the application responsible for that type of object. See **Opening Files and Their Applications** in Chapter 6—**ACCESSING APPLICATIONS**.

<Ctrl> When pressed simultaneously with another key, this modifies the sequence sent by that key. It is not used in the User Agent.

<Del> Interrupts ongoing operations. This is not used in the User Agent.

<Dlete> ASCII 'del'— Deletes the selected object(s) or text. Deleting a file moves it to the Wastebasket, from which it can be retrieved until deleted from the Wastebasket. Deleting a folder moves both the folder and its contents to the Wastebasket, from which the folder or any part of its contents can be retrieved. Objects that are deleted from the

Wastebasket cannot be retrieved; they are permanently removed. Delete cannot be used on an open file (unless it is a word processor); the user is prompted to close the file first.

<Dlete Char> Deletes the character *under* the cursor and closes the gap in text. It is an autorepeating key.

<End> Moves the cursor or highlight to the last character or field on the current line (text) or in the current window. <Shift>-<End> moves the cursor or highlight to the last character or field in the current object.

<Enter> This general terminator for all typed commands executes a form. It maps onto <B1> of the mouse. See **Process Control Commands: Conventions** later in this chapter.

<Esc> "Escapes" the meaning of the following keystroke. All hard keys on the **UNIX** PC keyboard are mapped onto escape sequences, as listed in the file **/usr/lib/ua/keymap**, so that users can simulate the use of hard keys as used by the User Agent and applications. This is needed for remote terminals that have different keyboards.

<Exit> Exits from the current application or window, closing the window. It is synonymous with close. See **Process Control Commands: Conventions** later in this chapter.

<F1> - <F8> These function keys have functions that can be assigned by each application. The names of the functions are displayed on the eight

corresponding screen keys, using a **TAM** call.

When shifted, these are dedicated keys used by the **UNIX** PC Telephone Manager.

<Find> Finds an object that matches user-supplied search criteria, and points to it or displays a list of objects meeting the search criteria. The user is prompted to supply the search criteria in a form. The User Agent Find command displays the matching objects in a menu. The user can select objects from the menu and issue commands on them. Pressing the <F6> key executes this command.

<Help> Gives context-specific help on the selected object, action, or the current message, and gives the user access to the help system. If context-specific help is not available, it gives the user access to the help system.

<Home> This is the same as unshifted <Beg>. <Shift>-<Home> is the same as unshifted <End>.

<Input Mode> Toggles between insert mode and overtype mode, in editing. The top border of the window should display IN for insert mode and OT for overtype mode, where these modes are supported.

<Mark/Slect> Selects one or more objects from a list or menu; it begins selection of an area in text. It steps through the options in a constrained field and is equivalent to pressing <B3> on the mouse.

<Move> Moves the selected object or region of text to another location, and deletes the original. After specifying the object and Move, the user is prompted to point to the destination location and give the PASTE command (see **Other Commands** following this section). The destination window is updated immediately; the source window is updated the next time it is made active.

Move cannot be used to overwrite an existing file, like **mv** in the **UNIX** System V shell. If a file with the same name exists in the target location, the user is prompted to supply a new name for the file to be moved. When applied to a folder, both the folder and its contents are moved to the new location.

<Msg> Gives the user access to any asynchronous messages that have arrived. If only one type is queued, it invokes the utility or application needed to display the messages. If more than one type is queued (for example, system messages and electronic mail), it displays a menu to allow the user to choose the type of message to be viewed. It is a dedicated key and cannot be used by applications. This command is not available on remote terminals.

<Next> Each keystroke moves the highlight or cursor to the next unit. In the User Agent, it moves the highlight to the next item in a list or the next field in a form. It is an autorepeating key.

<Num Lock> Switches keys in the numeric pad on the lower right of the keyboard between numeric pad and cursor control functions.

Keys are translated as numbers when the light in Num Lock is on; keys are translated as cursor commands when the light is off. See kbd(7).

<Open> Opens the marked or named application that owns the object for editing or viewing in a new window. It is the default command for most objects in the User Agent. Open is the only command that can be used on objects in the Office window.

<Opts> Displays the available options on a command or in a field on a form. It might display a menu or a form, to allow the user to specify choices. In User Agent, it is used only in forms to allow users to see the options that are available in fields whose contents are restricted.

<Page> Displays the next windowful of the text or the menu that is contained in a window. <Shift>-<Page> displays the previous windowful of text or menu.

<Prev> Moves the highlight or cursor to the previous unit. In the User Agent, it moves the highlight to the previous item in a menu or the previous field in a form. It is an autorepeating key.

<Print> Sends the selected item to a printer. It displays a form that allows the user to select the printer, and allows the user to specify other options such as number of copies. In the User Agent, the print command is executed directly for certain file types, such as standard files and phone directories; for other file types, the application must be invoked. The User

Agent will invoke the application to perform printing if **print** is defined for the object in the Suffixes file. See **Printers** in Chapter 10—**CONTROL OF PERIPHERAL DEVICES**.

<Shift>-<Print> prints a bit-by-bit snapshot of the screen, if a dot matrix printer is available. It is a dedicated key and cannot be used by applications.

<Redo> Re-executes the previous command, allowing the user to respecify arguments or edit the command line. It is used in the User Agent.

<Ref> Gives access to an on-line reference facility. It is not used in the User Agent.

<Reset> Performs a soft reset. It is not used on the **UNIX** PC.

<Return> General terminator for all typed commands ; moves the highlight to the next field in a form.

<Rfrsh> Refreshes the screen.

<Rplac> Replaces an old object or piece of text with a new one. It is not used in the User Agent.

<Rstrt> Takes the user to the entry level of the current task. It is not used in the User Agent.

<Rsume> (1) Suspends the current window or application and displays the window manager menu. (2) Within the window manager menu, it resumes the window that is under the highlight. It is the default command within the window manager menu

and is a dedicated key not available for use by applications. This command is not available on remote terminals.

<Save> Saves changes to current file, with the option of saving under a new name. An application must verify that no existing file in the current directory has the new name to prevent accidental overwriting of an existing file. It is not used in the User Agent.

<Suspd> Suspends the current window or application and displays the window manager menu. It is a dedicated key not available for use by applications. This command is not available on remote terminals.

<Tab> Moves the highlight to the next field or tab stop. In text creation, it inserts a tab character.

Shifted, it moves the highlight to the previous field or tab stop.

<Undo> Undoes the result of the last executed command. It is not used in the User Agent.

Other Commands

Other commands used in the User Agent for File management, window management, and system control are listed below. These commands are available on the command menu and some are also available on screen keys. Many of the commands found on the screen keys are also found on the file system commands menu (displayed by pressing <B1> or <Cmd>).

Cleanup Available in the Office and file system commands menus. It closes all open

windows owned by the User Agent except the Office.

Logout — This is in the Office Commands menu. It logs the user out. Users must close all open files before logging out.

Move — This command appears in the Window Commands menu. It activates the highlighted window and allows the user to move the window without the mouse by using the arrows on the keyboard.

NEXT WINDOW — This is on screen key <F5>. It activates the next (algorithmically determined) window. It only activates Office, file system, Administration, Floppydisk, Clipboard and Wastebasket windows. It does not activate application windows, or any windows that do not display the NEXT WINDOW and PREVIOUS WINDOW screen keys.

ORGANIZE — This is on screen key <F1> in User Agent windows. It allows the user to specify the display format for the current file system window, to control the amount of information about each file or folder that is displayed, and to control the format in which it is displayed. It takes no object.

PASTE — This is in the file system commands menu and on screen key <F8> in User Agent windows. It completes a Copy or Move command. When the user has selected a file or folder and given the Copy/Move command, the user then opens the folder in which the object is to be placed and presses <F8> or types "<Cmd> paste."

PREV WINDOW — This is on screen key <F4>. It activates the previous window and is subject to the same restrictions as <Next Window>.

RENAME — This is on screen key <F3> in File System windows and in the Commands menu. It displays a form that allows the user to type in a new name for the selected object. The user does not need to supply a suffix; it is automatically appended to the name. Rename cannot be used to overwrite an existing file, like **mv** in **UNIX** System V. If an object with the new name already exists, the user is prompted to specify a different name.

Run — This is in file system command menus. It executes the program contained in the specified object and is the default command for executable files.

SECURITY — This is in file system command menus and on screen key <F2>. It is a User Agent command that allows the user to set access permissions and ownership on the specified object.

Shape — This is in in the Window Command menu. It activates the window under the highlight and allows the user to change its shape by using the arrow keys on the keyboard.

SHOW LAYOUT — This is in the file system command menu and on screen key <F7>. It displays the hierarchical organization of the user's file system pictorially from the current window down.

Shutdown	This is in the Office and file system command menus. It checks to determine if any remote users are logged in and if mail is being sent or received. It asks for confirmation before proceeding, then it synchronizes (**sync**s) the system, logs the user out, and prepares the system to be turned off or rebooted.
System info	This is in the Window Manager Command menu. It displays information about the current active window and is provided as a utility for software developers.

Process Control Commands: Conventions

This is a detailed description of the consequences of pressing the keys that control execution, cancellation, and termination of functions and applications, for each active window type. These functions should be used consistently in all applications.

Figure 1-5 shows the results of pressing the <Exit>, <Close>, <Cancl>, <Enter>, and <Return> keys, and of pressing <B1> on the [X] icon in the window border, on the [OK] patch in a form, or when not pointing to an icon, for different active window types.

	ACTIVE WINDOW				
KEY/PATCH	*popup output only*	*"Must select" menu*	*"May select" (persistent) menu*	*Form*	*Application window*
EXIT key	close popup window; query	cancel cmd/ selection (no change); close	cancel cmd/ selection (no change); close	cancel initiating command; no change to form; close form window; from calling window. exit to node	cancel cmd/ selections query to save changes → close application window
CLOSE key	save changes→ exit to node	menu window; query to save changes→ exit to node	window; exit to node		
CANCL key	close popup window; return to calling window	cancel cmd/ selection (no change); close menu window; return to calling	cancel cmd/ selection (no change); leave window open	cancel initiating command; no change to form; close form window; from calling window. exit to node	cancel cmd/ selections; leave window open
ENTER key	close popup window; return to calling window	take action or select item under high-light; close menu window return to calling or specified window	take action or select item under high-light; leave window open	take action, send form and close form window; return to calling or specified window	defined by application
RETURN key	close popup window; return to calling window		take action or select item under high-light; leave window open	move highlight to next field	defined by application
B1 or ☒	close popup window; return to calling window	(no change); close menu window; return to calling window	selections; leave window open (2) if no cmd/ selection in effect, close window, exit to node	cancel initiating command no change to form; close form window; return to calling window	(1) cancel cmd/ selections; leave window open; (2) of no cmds/ selections in effect, query to save changes →close application window

Figure 1-5. Definitions of Execution, Cancellation, and Exiting Functions, by Window Type (1 of 2)

	ACTIVE WINDOW				
KEY/PATCH	*popup output only*	*"Must select" menu*	*"May select" (persistent) menu*	*Form*	*Application window*
B1 or OK	close popup window; return to calling window (patch not needed)	not available	not available	take action, send form, and close form window; return to calling or specified window	defined by application
B1 not on an icon	close popup window; return to calling window	item under highlight; close menu window return to calling or specified window	item under highlight; leave window open	select field and cursor position within field	defined by application

Figure 1-5. Definitions of Execution, Cancellation, and Exiting Function, by Window Type (2 of 2)

Notes On Usage

- The <Cancl> key cancels a partially specified command and unselects any selected objects. A popup window contains a form or menu that results from issuing a command, such as an options sheet for a printer, and can be considered to be a part of the command. Thus, cancelling the form should result in 1) leaving the initial settings in the form unchanged, 2) closing the window containing the form, and 3) returning to the environment from which the command was issued.

- Generally speaking, the <Exit> key is used to terminate an application, even if another window owned by that application, such as a command menu, is open at the time. The Exit command should result in a query as to whether the user wants to save any unsaved data. Following response to the query, the application and its associated windows are closed, and the "previous" window is made active.

It is not always desirable to exit from an application completely. For example, when the user gives the Exit command from the Phone Manager's "Edit directory" function, the *intent* of the command is usually to return to the main Directory screen, not to exit from the application completely. Rather than applying a rule about the use of the Exit command, exit should be used as a "do what I mean" command. This means that developers should assign its consequences on a case-by-case basis to avoid taking too large a jump through the menu hierarchy when the command is issued.

The following rough guidelines can be used:

- If the Exit command is issued from a "sub-application" within an application, exit back only as far as the menu or environment from which the sub-application was generated.

- If the Exit command is issued from a simple menu that is not a part of a sub-application, then exit from the application as described above.

In Figure 1-5, the notation, "exit to node" is used to convey the above use of the Exit command.

- The [X] patch in the window border is mapped onto the <Shift>-<Cancl> key on the keyboard.

- A function to allow an [OK] patch to be turned on and read is provided within the body of a window containing a form. See Form(3T). The user points at the [OK] patch with the mouse and presses <B1> to execute the form. This is equivalent to pressing <Enter> on the keyboard. Pressing <B1> elsewhere in the form will select a field as the current field.

- In a popup menu, once a selection is made, there is generally no reason for the window to remain open, so it is closed as soon as the user makes a selection or cancels the menu.

  In persistent menus, which remain displayed after selections are made, the user must explicitly close (<Exit> or [X]) the window. Examples of persistent menus are all windows listing the file system contents and also the Administration menu.

Dedicated Keys

The following keys are *not* passed to application programs, but instead are passed directly to the kernel. Therefore, they are not available for use by applications.

<Msg> Used to access asynchronous messages (see **Asynchronous Messages, *Access Via Keyboard*** in Chapter 12—**FEEDBACK, PROMPTS, AND MESSAGES**).

<Suspd>, <Rsume>
Used to suspend the current window and invoke the Window Manager menu (see Chapter 5—**THE WINDOW MANAGER**).

<Shift>-<Print>
Prints the screen. It allows the user to direct a bit-by-bit snapshot of the screen to a dot-matrix printer (see **Printers, Printing the Screen**).

<Shift>-<F1>
These shifted function keys are reserved for use by the Telephone Manager to give immediate access to telephony functions from any environment. (See the *AT&T* **UNIX** *PC Telephone Manager User's Guide.*)

Chapter 5

THE WINDOW MANAGER

The Window Manager allows the user to manipulate all open windows. It displays a popup menu of windows to which the user can point and issue window shaping and movement commands. The Window Manager window is displayed when the user 1) presses <Suspd> to suspend the active window, or 2) points to the dotted background in the work area and presses <B1>, or 3) points to the [W] icon on the Status line and presses <B1>.

The Window Manager lists the following information about each window:

- Object name: the name of the object whose contents are shown in the window.
- Application name.

The list of windows is a menu. When the user displays the Window Manager window, the menu item corresponding to the just-suspended window will be highlighted. The user selects a window by pointing to an entry in the menu.

Pointing at an entry in this menu and pressing <B1> or pressing <Enter> resumes the window at which the mouse pointer is pointing. Pressing <B2> displays a menu of commands to act on the selected window.

The commands available are: resume, shape, move, and system info. Resume is the default command. Pointing at [X] and pressing <B1>, or pressing <Exit>, resumes the previously active window.

Controlling Windows

The user can control the location and shape of a window with the mouse as described in **Window Border Areas** and **Default Window Properties** in Chapter 2—**DISPLAY**.

Navigating Between Windows Using The Mouse

The current window is implicitly suspended by moving the mouse pointer outside the current window and pressing a button. The process that becomes current depends on the location to which the mouse is pointing when the button is pressed:

- If the mouse points to any visible part of another window and any button is pressed, the current window is suspended, and the window pointed to is resumed.

- If the mouse points to the telephone region of the status line and any button is pressed, then the current window is suspended, the Telephone Manager is invoked, and the Call Screen is opened and made active.

- If the mouse points to an icon in the message region on the status line and <B1> is pressed, the current window is suspended and the first notice is displayed in a popup window or the corresponding service is invoked, depending on the region. The interaction of the mouse with the message region is described in **Asynchronous Messages, *Access Via Mouse*** in Chapter 12—**FEEDBACK, PROMPTS, AND MESSAGES**.

- If the mouse points to the dotted background in the work area or to the [W] icon on the Status line and any button is pressed, the Window Manager is displayed in a popup window.

Chapter 6

ACCESSING APPLICATIONS

This Chapter describes how files are opened in applications and how data is transferred between them. From the user's point of view, these operations are performed through the User Agent. Applications are not explicitly invoked. Instead, the object they operate on is opened or created, and the application is automatically loaded to operate on the object. Applications that cannot be invoked in this way should install themselves in the Services menu in the Office (and install a Services menu if none exists). Users then invoke the application from the Services menu.

Opening Files And Their Applications

The user chooses a file on which to work, and then opens it using one of the methods described in Chapter 2—**DISPLAY**.

Opening a file invokes the appropriate application to edit or display the object, and passes the name of the object to the application. The application opens a window and displays the file in the window.

The application that is invoked depends on the hidden suffix on the filename. The effects of Open and all other commands on the objects with this suffix are defined in **/usr/lib/ua/Suffixes** [see ua(4)]. An entry is placed in this file for each object type owned by the application. This is done as a part of the installation procedure. See Chapter 5—**SOFTWARE INSTALLATION** in Section 2.

The user can create a new object, such as a spreadsheet, by using the Create command. The user opens the window in which the object will be stored, and issues the Create command.

(If the object cannot be stored in the current window, for example, the Office, it is created in the Filecabinet and the user is so informed.)

Create presents a menu of types of objects that can be created, which is taken from the **/usr/lib/ua/Suffixes** file. The user selects an object from the menu and presses <Enter>. The user is then prompted to enter the object's name in a form. When the form is executed, the appropriate application is invoked on the newly created file.

Exiting From an Application

An application window is closed when the application is exited. The exit can be by the application's standard command or by the user pressing the <B1> mouse button on the [X] in the window border. (Clicking on the cancellation region, [X], should close the window only when there is no current selection in the application. If an object in the window is selected, clicking on the cancellation region should cancel the selection.) See tam(3T).

When the command to close the window is issued, the application should prompt the user to make certain that no work is lost (for example, prompt to save document changes).

The window is closed and removed from the display. Windows or portions of windows that were under the window are refreshed.

Data Transfer Between Applications

Two methods are available for transferring data between files of the same or different type. (The applications involved must support a common intermediate file format.) The first method allows the user to move or copy a block of data between windows without explicitly moving the data to an intermediate buffer. The second method uses a set of special files on the

Clipboard in the Office. These files are created by copying or moving data from a file, which is currently running under an application. In both cases, conversion of the data to a common format is performed. The structure of the data (for example, spacing and tabs) should be preserved, so that the user is not required to reformat the data in the new file. See dif(4).

Method 1: The Invisible Clipboard

Moving or copying data from one file to another is very similar to moving or copying data within a file. The user gives the application's move or copy command, and specifies the object or region to be moved or copied. The item to be moved or copied is highlighted. A region is highlighted using the commands that normally specify regions in the current application.

To specify the destination of the material, the user moves the cursor to the target window, by opening or resuming the target file, points to the position where the material is to be placed, and gives the Paste command (or the application's version of this command.) The unnamed buffer is *not* flushed when the user pastes its contents in a new location. This allows the user to make multiple copies of the same material. The buffer contents are replaced when the user issues a new Move or Copy command.

When the Move or Copy is completed, the target file is updated to reflect the result of the Move or Copy.

Method 2: The Clipboard

The Clipboard in the Office can contain an unlimited number of files which are used as repositories of data being transferred between files. The availability of more than one Clipboard file allows the user to move or copy several objects or regions from the source file before opening the target. Clipboard files can be retained indefinitely.

They are useful for storing frequently used data or text, such as headers for internal memoranda.

When transferring data **to** a Clipboard file, the user highlights the region or object to be moved or copied and issues the Move or Copy command. To specify the target, the user opens the "Clipboard" from the Office window and gives the Paste command. The user is then prompted to type a name for the data to be stored on the Clipboard, and it is stored under that name.

When moving or copying data **from** the Clipboard, the Clipboard file is the object of the command. The object is specified by opening the Clipboard and pointing to the desired file, and pressing Copy or Move. The user then opens or resumes the target file, puts the cursor in the target location and gives the Paste command.

Clipboard files can be copied, moved to a file, or deleted but not opened. To view a Clipboard file, the user must copy or move a Clipboard file to an open file in a window and paste it in that file.

Interaction With the File System—Bringing in Files

To copy the contents of a file into an application window, when an application allows it, the user does the following:

- Suspends the application
- Moves to a file system window
- Highlights the file name
- Issues the Copy command
- Resumes the application and moves the cursor/pointer to the target point within it

- Gives the Paste command.

The file is then read into the current file.

Chapter 7

MENUS AND FORMS

This Chapter describes menus and forms. Menus and forms display available commands and objects to the user. The use and design of menus and forms used in the User Agent is described.

Menus

A menu gives the user a way to view a list of available items and allows the user to select:

- an item from the list for execution (action)
- one or more objects for an action
- an option on a command, or
- an entry in a restricted field in a form.

See **Command Entry Methods** in Chapter 3—**INPUT** for a complete description of how items are selected from menus. Also, see menu(3T).

The menu system supports the following functions:

- Selecting and highlighting the name of the object under the cursor, described below.
- Highlighting the names of all selected items.
- Allowing the user to move the highlight from item to item using the cursor control keys or mouse.
- Scrolling and paging the menu in the window vertically, if the menu is too long to be displayed in the window.

- Canceling selected items via the <Cancl> key or a region in the window border.

- Multiple selection or single selection. Some menus allow simultaneous selection of several items (for example, the Filecabinet menu). Others allow only one item at a time to be selected (for example, a command menu). The choice of the type of menu depends on the context.

Selection Indicators

Selection indicators are used to indicate the menu items(s) that will be selected or affected by an action. Selection indicators cover the entire name of the object.

Movable Highlight

The *movable* highlight is an inverse video bar with a "-" to the right of the menu item that indicates the "current item." The highlight covers the entire item. If the user issues a command, the "current item" under the highlight becomes the item on which action is taken.

As described in **Selection by Pointing**, the highlight is moved in menus one of three ways: using the mouse pointer, using the keyboard arrow keys, or by typing. The highlight follows the mouse pointer when the tip of the arrow is in a menu and pointing to an item (or within a "capture region"—a 1-character-position region to the right and left of the item).

The highlight only moves in the active window. When the user has selected an item by pressing <Mark> or <Cmd>, that item will remain highlighted while the highlight continues to follow the mouse pointer. This is described in more detail next.

Stationary Highlight The *stationary* highlight is an inverse video bar that covers the entire item name and does not move with cursor arrow or mousetrack. It is a "sticky marker" that is turned on when the user selects an item for future action by pressing <Mark> or <Cmd>. More than one item can be selected. If the user specifies an action, all explicitly selected items are passed as arguments to the action. If **any** item has been explicitly selected, the item under the moving highlight is not passed as an argument unless it also has been explicitly selected. See **Selection by Pointing** in Chapter 3—**INPUT**.

The highlight is turned on/off when an item is under the pointer marker and the user presses <Mark> or the <B3> on the mouse. The user will not recognize that the highlight has been turned off until the mouse is moved away from the item.

Stylistic Considerations for Menus

Menus are normally alphabetized. If a menu contains two or more columns, the contents are alphabetized by column, not by row. This makes a big difference in the ease with which the user locates an item in a menu. See menu(3T).

Items in menus should be all lowercase, or mixed upper and lowercase, for readability. All uppercase characters should not be used.

The label area of the menu window may contain an identifier for the menu. This label identifies the contents and should reflect the command or situation that invoked the menu.

The first line in the menu window may display more information about the contents of the menu. A single prompt can accompany the menu, and it is displayed on line 26.

Context-sensitive help can be attached to each item in a menu. That is, a different help screen can be displayed, depending on the item highlighted.

Forms

Forms are used for command entry when a command contains user-specifiable options. A form will be presented for the user to specify options for running the command. Forms are also used to display and allow the user to set user preferences and the properties of objects. They are used in many administrative functions. Forms may be created within a C program [see form(3)] or from the **UNIX** System V shell [see shform(1)].

Navigation in a Form

The user selects and highlights an unprotected field:

- by moving the mouse pointer over the field and pressing <B1>
- by pressing <Return> or <Tab> or <Next> to select and highlight the next field
- by pressing <Shift>-<Tab> or <Prev> to select and highlight the previous field, or
- by using the up and down arrow keys to move between fields.

The right and left arrow keys are used to position the cursor within a field.

Field Types

Forms contain two kinds of unprotected fields, *free-entry* fields and *constrained* fields.

Free Entry Fields

When the user positions the highlight on a free-entry field and begins to type, the previous or default entry is cleared from the field and the new entry is echoed as it is typed. If the first keystroke is an arrow key or an edit key, the field is *not* cleared when a character is typed. In this latter case, it is assumed that the user wants to edit the field contents, so the character is inserted in the text instead of the text being cleared.

Other editing functions include:

- BACKSPACE—deletes characters to the *left* of the cursor and closes the resulting gap
- DELETE CHARACTER—deletes the character *under* the cursor and closes the gap
- CLEAR LINE—clears the field containing the cursor
- INSERT MODE—characters are inserted at the left side of the cursor as they are typed, pushing characters that are under and on the right of the cursor further to the right
- OVERTYPE MODE—the character under the cursor is replaced with a typed character.

Insert and overtype modes are toggled by pressing <Input Mode>. "IN" and "OT" should be echoed on the top window border of the form to reflect these modes.

Constrained Fields

Only certain system-defined entries can be placed in a constrained field. These are defined in the calling program. The user can view the selections by pointing at the field and:

- Pressing <Cmd> or <Opts> or <B2>. This displays a menu of available selections for the field. When the user points at a selection in this menu and presses <B1> or <Enter>, the menu is closed and the item is displayed in the field on the form.

- Pressing <B3> or <Mark>. Each time the key is pressed, the next item in the options list will be displayed in the field. This list is circular, that is, the user cycles through all the items in the list, continuing with the first after the last.

- Beginning to type. The menu of available selections is displayed, as above.

Help In Forms

The calling program can attach a single-line (70 character) prompt to each field in the form. The prompt is displayed on line 26 whenever its field is active.

General help can be defined on a form. Help on a form is obtained by pressing <Help> or pressing <B1> on the [?] icon in the window border.

Exiting From Forms

To execute (save) the contents of a form, the user presses <Enter> or points to [OK] and presses <B1>. This executes the underlying command or saves the contents of the form, and closes the window on the form.

To cancel without changing any values and close the form windows, the user points to [X] and presses <B1>, or presses <Exit> or <Cancl>.

Layout of Forms

The top border of the form window displays the name of the menu item or command used to invoke the form. The top line of the form can display either more information or a single-line, form-specific prompt. It should reflect the name of the function used to call the form.

Forms should be laid out neatly, with field boundaries lining up where possible. Field names should be right-justified, and input fields should be left-justified. See the Office Preferences form for an example.

Mixed upper and lowercase characters should be used in field names, for readability.

If a field entry is constrained in length, the highlight should span the maximum number of characters and no more.

Error messages for invalid entries in fields should be specific about the nature of the problem; for example,

> File name: abc def
>
> Should return:
> A file name cannot contain a space.

On return from an error, the field in which the error occurred should be highlighted.

Chapter 8

CUSTOMIZATION

This chapter describes the features of the User Agent that can be customized by the user. The access to customization functions is via the Preferences menu in the Office. Opening Preferences results in a menu of areas that can be customized. In the Foundation software, these areas are *Office* and *Phone Manager*. Applications can install their entry points for customization here, as well as in the **/usr/lib/ua/Preferences** file. Selecting an area to be customized results in display of a form, which the user fills in. The form contains defaults or current values for each item that can be customized.

The following items are available for customization of the user's interaction with the **UNIX** PC and Office:

- *Standard window size*

  These values apply to standard files. Applications could also provide a choice.

- *Display of a command menu*

  Users can turn off the display of the command menu when <Cmd> is pressed. Applications can also provide this choice.

- *Display of Filecabinet and Folders*

  Users can select the default amount of information displayed about each object (name only, name and object type, or name, type, modification date, and size). They can select the order in which the objects are displayed (alphabetical, largest or smallest first, oldest or newest first, or alphabetical by object type).

For each folder, this display can be modified with the Organize command.

- *Access permissions of files and folders*

  Users can specify default access permissions for their own files and folders. These permissions can be modified for individual files and folders with the Security command.

- *Emptying Wastebasket*

  The user may elect to have the Wastebasket emptied automatically daily, weekly, monthly, or never.

Chapter 9

ACCESSING THE UNIX SYSTEM: EXPERT AND STANDARD MODES

This chapter describes the two modes with which users can access the **UNIX** System V: Standard mode and Expert mode. These modes differ in the user's ability to directly access the **UNIX** System V shell. This chapter describes the specific features of each mode of access.

Two modes are provided, *Standard* mode and *Expert* mode. These modes are set via Administration in User logins, under the "install" login only (install is the system administrator). These modes govern whether or not the user is allowed to break out of the User Agent environment and gain access to the **UNIX** System V shell or another **UNIX** shell. This is done primarily to prevent the user from accidentally "falling into the **UNIX** System V shell and getting lost or inadvertently modifying the file system.

In the Standard mode, the user is fully protected from "falling" into the **UNIX** System V shell and inadvertently issuing **UNIX** System V commands. From Standard mode, the user can type User Agent commands (*Typed Command Line Entry*) and commands that are recognized in programs or applications called by the User Agent. The user cannot use full regular expressions in typed commands or searches, but can use the wild card character "*." **UNIX** System V is not listed in the Office of the Standard User.

In Expert mode, the user has access to the same User Agent commands and applications provided in Standard mode, and can also run the **UNIX** System V shell (or any other installed shell) in a window and can issue shell commands in that window.

The Expert can run the shell by selecting **UNIX** System from the Office or by typing **!sh <Return>** from the Office, Administration, Wastebasket, or any file system window. A window will then be created, and the shell will run in that window. The user can create more than one **UNIX** System window. The working directory is the directory from which the shell command is given, or **/u/userid/Filecabinet**, if the command is given from the Office window. The user will be informed to **Press Ctrl d to exit** on the message line. The user returns to the User Agent by suspending the window or using the shell exit command.

The Expert user can also run single shell commands by typing **!command <Return>**, for example, **!ls -l <Return>**, from the same windows just mentioned. A window will be created to display the output of the command, if any, but no further input will be accepted into the window unless the command expects input. The user is prompted to **Press ENTER to continue** on the message line when the command has finished.

The Expert user can use some of the file management capabilities and commands of the User Agent on any **UNIX** System V directory. Any **UNIX** System V directory can be displayed in a window as a menu. A window will be opened on the parent directory of the current directory if the user types **.. <Return>p**. This method can be used to navigate the user's Filecabinet as well as other parts of the **UNIX** System Vfile system. The user can also open any directory in menu form in a window by typing its path name, for example, **/etc <Return>**.

When a directory is displayed as a menu in a window, any User Agent file manipulation command, such as Copy, Move, Delete, Rename, and so on, can be used. These commands provide a protection against accidental deletion and overwriting that is not available in standard shell commands. The user can also Open directories in their own windows and Open standard files in the default editor if permissions allow it. The user can execute a command (Run an executable file) by pointing at the

filename and pressing <Enter> or <B1> if the command requires no arguments and permissions allow it.

Chapter 10

CONTROL OF PERIPHERAL DEVICES

This chapter describes how the User Agent controls two peripherals: Printers and Floppy Disks.

Printers

Users can print an object by selecting the Print command (or <Print>) and print the screen by pressing <Shift>-<Print>. A form containing options, including defaults, is displayed.

Printing an Object

From the User Agent, the object of the Print command is specified by pointing to the object name on a menu or typing the name.

ASCII files and certain other objects, such as saved phone directories, can be printed directly from the User Agent. When selected for printing, the User Agent displays a form allowing the user to specify the printer and number of copies.

Printer Queue Control

The Printer Queue in the Office Printers menu allows the user to display a list of jobs queued for each printer, and to point to those to delete from the queue.

Printing the Screen

The screen can be printed by pressing <Shift>-<Print>. This command sends a pixel-by-pixel snapshot of the video display to a dot-matrix printer. This can be done with the sprint(1) command. The user cannot print the screen of a remote display.

Floppy Disk

From the Office users can access **UNIX** System V files and folders on a floppy disk, and copy and move them between the hard disk and the floppy disk; with the same user interface as copying and moving objects between the Filecabinet and other file system windows.* The floppy disk must be formatted, and it must contain a mountable file system.

A floppy disk must be mounted before it is accessed. The user does not need to mount a floppy disk explicitly, however. Instead, the user issues the command to Open the Floppydisk object in the Office. A menu of procedures then appears. Once the user makes a selection, he or she is prompted to insert the floppy disk and close the latch. Mounting the floppy disk is done transparently.

When a floppy disk is mounted, its contents (top level) are displayed as a menu in a file system window called Floppydisk. The menu looks identical to the Filecabinet menu. When a floppy disk is currently mounted, the red light on the disk drive is on.

---

* Other **UNIX** System V operations involving floppy disks are formatting, duplicating, repairing a damaged File system, installing applications, backing up and restoring. **MS-DOS** operations available from this menu include initialization, reading from and writing to an **MS-DOS** diskette. These functions are described in the AT&T **UNIX** PC Owner's Manual.

To unmount the floppy disk, the user must Close the Floppydisk window and all windows opened from the Floppydisk window. This syncs and dismounts the floppy disk. All synchronization and dismounting is done transparently.

If the user flips the latch on the disk drive without first closing the Floppydisk window, a popup window appears. The user is prompted to close the latch, then close the Floppydisk window before removing the floppy disk.

Chapter 11

HELP SYSTEM

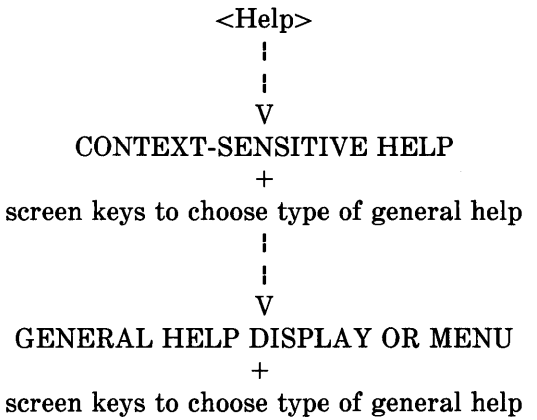
This chapter describes the User Agent's Help system. The structure of the Help system should be followed by applications. Two types of help are available to the user:

1. Context-specific information (which gives access to the following)

2. A general Help service providing access to various types of information.

Context-Sensitive and General Help

Context-sensitive help and general help are accessed through the Help command or by pointing the mouse pointer at the [?] icon in each window border and pressing <B1>. The following example illustrates levels in the Help system.



Context-Sensitive Help

When help is requested, the first help information that is displayed consists of a window containing a brief message which may be scrolled. The content of the message depends on the application, the environment, and the position of the cursor.

- If the user is filling out a form, the message should provide a definition of form and gives information about what is required in each field.
- If a menu item in a file system window is highlighted, the message should provide information about objects of this type.
- If a menu item in a command menu is highlighted, the message provides information about the function and syntax of the command.
- From an error window, the message should provide more information about the error and how to correct or avoid the error.

The window remains until the user explicitly closes it.

General Help

From the context-sensitive help window, the user can access the general Help service. The screen keys displayed while the initial Help window is active provide the user with several entry points to this service. These screen keys are displayed continuously while any help window is active; they help the user navigate through the Help system. Each screen key takes the user to a display of information or to a menu.

The first two screen keys should be:

Table of Contents <F1>	Displays in menu form a listing of all help screens. The user can select and display screens of choice from this menu.
Using Help <F2>	Gives a brief description of how to use the help system.

The other six screen keys should be related major topics for which the user can request more information. When the user displays one of these screens, other screen keys related to that screen can be displayed. See uahelp(4) for more information.

The Help messages should be self-explanatory and should not assume that the user has read other Help messages.

Help Display

When the user specifies the type of help desired, the label line of the window indicates the nature of the help provided. At the bottom of each windowful of information is an indication of whether more information can be seen by scrolling. Users can scroll or page forward and backward in each help display.

Navigation and Exit From Help

The user can move to different parts of the Help service using the screen keys. The user can close the Help window and return to the previous context by pressing <Cancl> or <Exit>.

Chapter 12

FEEDBACK, PROMPTS, AND MESSAGES

This chapter describes the types of feedback given to the user. The following information does not apply when the user issues a **UNIX** System V command or exits into the **UNIX** System V shell.

Command Echo

Many commands and the name of any currently selected object are echoed when entered, so the user knows with certainty what was requested. If the user typed a wild card character in a command, the wild card should be expanded when the user completes the command. If several objects are selected, their names should be echoed when the action is specified. Echoed commands should be displayed long enough to be read (at least 2 seconds), but should be cleared when no longer in effect. See window(7).

Prompts

Prompts provide information to aid the user in performing certain actions. (For example, when the Move command is given, the system provides a prompt to the user that it is now necessary to point to a destination for the object to be moved.) These prompts should be provided whenever invoking an action requires more than just pressing a function key. Prompts are displayed on line 26. They should be cleared when they are no longer applicable to the current context. See window(7).

Informational Feedback

A message should be given to the user confirming that his/her action has had its expected effect, or that the system is processing the request. These messages are needed when there is no other indication to the user that an action has had an effect. These messages appear on line 26 if they refer to a foreground process. If they refer to a background process, they are displayed as asynchronous messages, as described below. Messages should be cleared when the user changes the current context.

Error Handling

The following means are available to signal to the user that a keystroke is invalid, a system error or failure occurred, or that a dangerous condition exists. See message(3T) and message(1).

- Do nothing
- Beep
- Display a status manager icon [!!]
- Display an error message in a popup window
- Display an error message on the message line.

In general, use of the beep should be minimized; the beep should not be used to signal every possible error condition.

Use the following guidelines for each type of error message:

- Do nothing for minor errors, such as navigation errors. Examples:

  — Cursor/scroll/page up when at the top of the list/form/document

  — Cursor/scroll/page down when at the end of the list/form/document

- Next/Prev window when the only User Agent window is Office.

- Beep alone, for some invalid keystrokes or commands:

  - Invalid keystroke—use *only* for keystrokes that are never used, such as Rstrt; do *not* use for functions that are sometimes available.

  - Pressing an unlabeled function key.

- Beep plus display a status manager icon:

  - Nonserious asynchronous error conditions, such as printer out of paper or not connected.

- Display a popup window alone, no beep or blink:

  - Use for commands that are sometimes available but not in present context, for example, Move Administration (Message: can't use Move on Administration), or Dlete Telephone

  - Use for commands that are not yet implemented (for example, Print foldername—gives message that the function is not yet implemented)

  - Nonfatal system errors (for example, an error occurred in dialing)

  - No help available

  - Nonserious user errors (Rename—file name already exists)

- Pop-up window plus three beeps (serious error with potential for damage and loss of data):

- Floppy disk latch opened while disk mounted
- Attempted write to incorrect floppy disk

- Display error message on the message line plus three beeps to signal a serious error where it is not possible to create a popup window:
  - Out of window resources—prompt to close some windows
  - Panic—prompt to record message and press reset button at the back of the **UNIX** PC.

Contents of Error Messages

Error messages shown in popup windows should be brief, but they should identify the problem and, where appropriate, point the user towards making a correct entry or correcting a problem situation. For example, **Printer unavailable** is not a useful message. Instead, the message or a help screen associated with the message, should instruct the user to check for the printer being connected, properly configured, plugged in, turned on, ready, and supplied with paper and ribbon.

Error messages should avoid the use of **UNIX** System V jargon, such as full path names and suffixes that are appended to file names.

At the end of popup error messages, the user should be told how to close the window, for example, **Press ENTER to continue**.

If help is available for the error message, the [?] icon should be displayed; otherwise it should be off.

Error messages should be small enough to fit into a window; no scrolling should be necessary.

Asynchronous Messages

Asynchronous messages include:

- messages sent by processes running in the background
- notification of incoming mail
- notification that the message center has a message for the user, and
- timed reminders.

An asynchronous message is stored when it arrives, and the user is notified that a new message has arrived. The notice that a message has arrived is displayed in the message region of the status line (line 1). Separate regions of the status line are defined for system messages, mail, and other notices. Each region remains blank until a notice corresponding to that region arrives. When a notice arrives, a beep sounds, and an icon is displayed. The icon is displayed until the user accesses the corresponding service or notice display.

Access Via Mouse

With the mouse, the user accesses asynchronous messages by pointing to the icon on the status line and pressing <B1>. The result depends on the region:

- *Mail*

  The mail service is invoked, if one is installed. Otherwise, the user is informed "you have mail" and must use **/bin/mail** to read it.

- *Notices*

  All other notices, including timed reminders and system messages from background processes, are displayed in a popup window, one at a time. Notices are displayed in last-in-first-out order.

Access Via Keyboard

All messages are accessed by pressing the <Msg> key. If only one type of message is present, the messages are displayed in a popup window, or the appropriate service is invoked. If more than one type of message has arrived, a menu of available message types is displayed, allowing the user to select the type desired. The messages are then displayed in a popup window or via the appropriate service.

Section 2

DETAILED INTERFACE SPECIFICATION

PAGE

Section 2

DETAILED INTERFACE SPECIFICATION

Chapter 1

INTRODUCTION

You should follow the guidelines described in this section (Section 2) for compatibility of your programs with the present release and future hardware and software enhancements of the **UNIX** PC.

UNIX PC Hardware Description

The **UNIX** PC is a desktop intelligent workstation using the Motorola 68010 microprocessor with the **UNIX** System V operating system. Virtual memory operation is provided. Minimum real memory is 512K (including memory for the operating system) and maximum process size for a virtual memory program is 2.5 megabytes. Note that the maximum is limited by swap space on the hard disk; for example, on a system with a 10MB hard disk the 3MB swap space is taken up when the system is booted. This leaves 1.5MB for all processes together. Much of the swap space is taken up by background processes, leaving little space for other processes.

Other hardware features include:

- 720 by 348 bit-map graphics screen
- DTE RS232-C serial port

- 36-pin, **CENTRONICS®**-compatible parallel printer port
- 103-key keyboard
- 3-button mouse
- AT&T PC 6300- and **IBM*** PC-compatible 48-TPI floppy disk drive
- Winchester disk drive (at least 10 megabytes)
- AT&T 212-compatible modem with auto-dial and auto-answer capabilities
- Two ports for connection to phone lines
- One port for connection to a telephone handset
- Three expansion ports.

Software Environment

The basic system software of the **UNIX** PC is the **UNIX** System V operating system. The User Agent, a user-friendly interface, is described in **THE *UNIX* PC USER INTERFACE** section of this guide. For applications to be fully compatible with the **UNIX** PC, they should work under **UNIX** System V **and** should conform to the specifications of this document and interface correctly with the User Agent. Programs that are **UNIX** System V-compatible and use only standard input/output (but are not necessarily integrated with the User Agent) can still run on the **UNIX** PC. They should be installed using the installation procedure described in Chapter 5—**SOFTWARE INSTALLATION**.

---

* Trademark of International Business Machines, Inc.

The **UNIX** PC supports a wide variety of industry standard printers, plotters, and other hardcopy output peripherals. For a complete list of output peripherals supported by the **UNIX** PC, consult your authorized distributor, AT&T representative, or your *AT&T* **UNIX** *PC Owner's Manual.*

UNIX System V Enhancements on the UNIX PC

The **UNIX** System V implementation found on the **UNIX** PC is certified to conform to AT&T Bell Laboratories specification published for **UNIX** System V. This implementation provides excellent source level portability from other **UNIX** System V implementations.

The implementation of **UNIX** System V found on the **UNIX** PC provides a number of features that allow software vendors to take advantage of the unique features of the **UNIX** PC. These features include record locking, shared libraries, bit-mapped screen with kernel support for multiple, asynchronously active, overlapping windows, a mouse, and an integrated modem. For further information on these features, see locking(2), shlib(4), window(7), and phone(7) in the **UNIX** *System V User's Manual.*

Software Development System

The **UNIX** PC software development system includes two different terminal virtualization packages, *Terminal Access Method* (**TAM**) and **curses**. Each provides device-independent terminal I/O.

The **TAM** package is recommended for programming on the **UNIX** PC because it is integrated with the operating system to make optimal use of the hardware. **TAM** has the following features that are not available in **curses**:

- The shared library feature of the **UNIX** PC is used, so programs written with **TAM** can be significantly smaller

than those written with **curses**. Programs written with **TAM** will take up less disk space and will load faster than programs written with **curses**.

- Real, overlapping windows are supported.
- Context-sensitive help messages are supported.
- Device-independent input is supported. (**curses** only supports device independence on output.)
- Menus, forms, and messages are supported.
- Both high- and low-level mouse support routines are provided.
- The most frequently used **curses** calls are emulated by **TAM** to allow easy porting of code already written using **curses**.

Programs previously written with **curses** can be ported using the **UNIX** PC **curses** package. The subset of **curses** supported on the **UNIX** PC is documented in curses(3X) (see the **UNIX** *System V User's Manual*). The remainder of this document describes the **TAM** package.

Output To The Screen

All access to the **UNIX** PC screen, windows, and remote terminals by application programs is via calls to the **TAM** package supplied by AT&T. **TAM** provides the hardware independence between application software and terminal screens and keyboards that is similar to what is provided by the **UNIX** System V **curses** package. **TAM** has the following features that are not in **curses**:

- Terminal-independent, low-level window primitives

- Loadable font control, including character graphics
- Support for bit-mapped graphics
- Low-level mouse control (reading and scaling)
- High-level window functions: creation, deletion, sizing/shaping, positioning.
- Menu/forms display and formatting, plus entry selection via keyboard or mouse
- High-level mouse control: auto-mouse tracking, cursor control, scaling, and button handling.

The **TAM** package is described in Chapter 3—**THE TERMINAL ACCESS METHOD** and includes facilities to:

- Position the cursor on the screen
- Write characters and strings to the screen
- Perform formatted output (**printf** functionality)
- Clear the screen or sections of the screen
- Insert and delete characters from the screen.

All applications should perform screen I/O through **TAM**. No direct calls to the hardware should be used. This allows future hardware enhancements to be transparent to application programs. Only **TAM** has to be modified to take the hardware enhancements into account.

When programming for the **UNIX** PC, you should refer to Chapter 3—**THE TERMINAL ACCESS METHOD** in this section and to tam(3T) in the **UNIX** *System V User's Manual.*

Alternate Character Sets and Character Graphics

Up to eight different character fonts per window can be used on the **UNIX** PC at the same time. For a program to use any character set other then the normal set, it must first define the alternate set on disk (or select one of the alternate font sets supplied), then it must load the desired set(s) into memory before using them.

If you want to define your own custom font, build a file as described in the font(4) in the **UNIX** *System V User's Manual.* The contents of a font file can be reviewed on the screen by using the cfont(1) command. The font file is then specified at font load time.

The procedure for loading the character set is described in window(7) in the **UNIX** *System V User's Manual,* and the loading is accomplished via an **ioctl** call with the WIOCLFONT option set.

Although each window can have up to eight fonts loaded at a given time, the font memory is limited to 64K. This means that if eight large fonts are defined, they cannot all fit in the font memory at the same time. You are responsible for insuring that the fonts to be loaded fit within the 64K memory limit.

Once the selected fonts are loaded, you have two different methods for switching from one font to another:

1. If only two fonts are to be used (system font in slot 0, and alternate font in slot 1), an ASCII "shift out" character (hex "0e" or octal "16") sent to the window will shift to the alternate font. An ASCII "shift in" character (hex "0f" or octal "17") sent to the window will cause a shift back to the system font.

2. If more then two fonts are to be used, the font you want to be the active font at a given point is explicitly selected by

sending a 5-character escape sequence to the window. This sequence is composed of an ASCII "escape", an ASCII "[", an ASCII "1", the ASCII number representing the font slot to be used (0 to 7), and an ASCII "m".

These escape sequences, as well as the entire range of escape sequences that can be used on a window, are shown in escape(7) in the **UNIX** *System V User's Manual.*

Note: Use of these escape sequences instead of **curses** or **TAM** calls can make the resulting code incompatible with remote alphanumeric terminals.

See Appendix B for a sample program that shows how to use both methods described above for loading and using fonts.

At the conclusion of use of any font other then the system-supplied font in slot 0, you should unload the font slots 1 through 7 (0 is reserved for the system font and should never be disturbed) to free up the system resources. Fonts are automatically unloaded when a process terminates. However, if the fonts are not unloaded until a process dies, memory is tied up unnecessarily.

The fonts shipped with the **UNIX** PC include the following fonts:

- PLAIN.I.E.12.A
- monitor.8.ft
- mosaic.8.ft
- special.8.ft

- system.8.ft
- system.r.8.ft

These font files are found in the directory **/usr/lib/wfont** and can be viewed by executing the **cfont** command with the path name of the font file as an argument.

Bit-Mapped Graphics

For applications to write bit-mapped graphics on the screen, use the **wrastop** calls. These are the **TAM** system calls that allow you to access the display memory. The memory management unit in the **UNIX** PC does not allow direct access to the display memory. It can only be accessed in the supervisor mode of the 68010 chip and only the**UNIX** system kernel can access supervisor mode. All other programs run in user modes. A description of the **wrastop** call can be found in the wrastop(3T) in the **UNIX** *System V User's Manual.*

High-level Graphics

Two graphics libraries are bundled with each **UNIX** PC: **GSS-DRIVERS*** virtual device interface (GSS VDI) and **GSS-TOOLKIT*** kernel system (GSS GKS). These libraries allow the programmer to create graphics routines which are device-independent. No licensing fee is required for the use of these libraries.

The GSS VDI library is a C language subroutine package which provides device-independent graphics input and output on the **UNIX** PC and its peripherals, including several popular printers, plotters, display devices, the mouse, and the console.

---

* **GSS-DRIVERS** and **GSS-TOOLKIT** are trademarks of Graphic Software Systems, Inc.

It implements the published ANSI specification of the Virtual Device Interface and enhances the portability of applications software.

The GSS GKS library is a C language subroutine package which provides device-independent graphics input and output on the **UNIX** PC. It implements the published ANSI specification for the Graphics Kernel System. It provides functionality beyond the GSS VDI library.

> ***Note:*** The use of the GSS VDI library may significantly increase the run-time of an application.

Input From the Keyboard And Mouse

You should learn the keyboard usage conventions on the **UNIX** PC before mapping the keys to your application needs. The philosophy of **UNIX** PC key use is given in Chapter 4— **DEFINITIONS OF KEYS AND ACTIONS** in Section 1 of this guide.

As with the screen output, all keyboard and mouse input should be via calls to the **TAM** package. The **TAM** routine **wgetc()** is normally used by application programs to obtain input from the keyboard. The value returned by **wgetc()** depends on the mode set by the last **keypad()** call. The default mode of the system has the **keypad()** flag at 0. In the default mode, **wgetc()** returns 7-bit codes, input keys return a 7-bit character for the normal QWERTY keys, and functions keys are mapped into escape sequences which result in a string being returned.

If you set the **keypad()** flag to 1, the **wgetc()** call will return 8-bit characters for any key pressed. You should use the labels shown in the following table for the values of the returned codes. A table of **/usr/include/kcodes.h** is provided to assign the actual values to the labels shown in the following table.

See tam(3T) in the **UNIX** *System V User's Manual* for more information on **keypad()** and **wgetc()**

Mouse input is through the **track TAM** calls which are described in the track(3T) in the **UNIX** *System V User's Manual* or WIOCSETMOUSE and WIOCGETMOUSE calls which are described in window(7) in the **UNIX** *System V User's Manual.*

FUNCTION RETURNED FOR EACH KEYSTROKE			
LEGEND	UNSHIFTED	SHIFTED	CONTROL PRESSED
CLR LINE	ClearLine	s-ClearLine	s-clearline
RSTRT REF	Ref	Rstrt	Rstrt
CREAT	Creat	s-Creat	s-Creat
SAVE	Save	s-save	s-save
UNDO	Undo	s-Und	s-Und
REDO	Redo	s-Redo	s-Redo
FIND	Find	s-Find	s-Find
RPLAC	Rplac	s-Rplac	s-Rplac
MOVE	Move	s-Move	s-Move
COPY	Copy	s-Copy	s-Copy
DLETE	Dlete	s-Dlete	s-Delete
DLETE CHAR	DleteChar	s-DleteChar	s-DleteChar
SELECT MARK	Mark	Slect	Slect
INPUT MODE	InputMode	s-InputMode	s-InputMode
F1	F1	s-F1	-
F2	F2	s-F2	-
F3	F3	s-F3	-
F4	F4	s-F4	-
F5	F5	s-F5	-
F6	F6	s-F6	-
F7	F7	s-F7	-
F8	F8	s-F8	-
EXIT	Exit	s-Exit	s-Exit
MSG	Msg	s-Msg	s-Msg
HELP	Help	s-Help	s-Help
SUSPD	Suspd	s-Suspd	s-Suspd
RSUME	Rsume	s-Rsume	s-Rsume
OPTS	Opts	s-Opts	s-Opts
CMD	Cmd	s-Cmd	s-Cmd
CLOSE OPEN	Open	Close	Close
CANCL	Cancl	s-Cancl	s-Cancl
PRINT	Print	s-Print	s-Print
CLEAR RFRSH	Rfrsh	Clear	Clear
PAGE	Page	s-Page	s-Page

FUNCTION RETURNED FOR EACH KEYSTROKE			
LEGEND	UNSHIFTED	SHIFTED	CONTROL PRESSED
BEG	Beg	s-Beg	s-Beg
HOME	Home	s-Home	s-Home
END	End	s-End	s-End
PREV	Prev	s-Prev	s-Prev
ROLL	Up	RollUp	RollUp
NEXT	Next	s-Next	s-next
RIGHT (>)	forward	s-Forward	s-Forward
DOWN	down	s-down	s-down
LEFT	left	s-left	s-left
DEL ESC	Esc	Del	-
TAB	*	BACKTAB	BACKTAB
BACK SPACE	Backspace	Backspace	Backspace
RESET BREAK	Break	Reset	Reset
ENTER	Enter	Enter	Enter

* Defined by C Language

Eight function keys are on the keyboard, and room is provided on the screen for eight labels. The keys are referred to as "screen keys" or "screen labeled keys (SLKs)." The **TAM** curses package provides routines for applications to use in displaying labels on the screen key area of the screen and in reading the screen keys when they have been pressed by the user.

> ***Note:*** All shifted screen keys are reserved for use with the system software. Shifted screen keys are owned by the Telephone Manager at all times and are used for speed dialing from any environment. Applications must not use shifted screen keys.

You should review the **UNIX** PC conventions for screen keys before deciding on functions to assign to them in your application. See **Conventions for Screen Keys** in Section 1—**THE *UNIX* PC USER INTERFACE**, of this guide.

Each screen key can contain sixteen characters, arranged in two rows of eight. See **Dedicated Keys** in Chapter 4—**DEFINITIONS OF KEY AND ACTIONS** in Section 1 of this guide for a list of keys not available for use by applications.

Printer System

Because of the multitasking nature of the **UNIX** PC and its **UNIX** operating system, application programs normally send all printer output through the AT&T-supplied printer spooler (lp). Details on the operation of the spooler can be found in lp(1) in the **UNIX** *System V User's Manual.* It is strongly recommended that all software be written to use the spooler.

The only case where an application program interfaces directly to the printer device is when an application, such as word processor and business graphics applications, requires real time control of the printer. This direct interface is restricted to foreground printing only. (Foreground printing means the application has direct control of the printer device, as is done in single-tasking microcomputers.) When foreground printing is invoked, printing by the spooler is disabled.

To avoid a conflict between this foreground printing capability of an application and the spooler, any applications using this foreground printing feature are required to:

- Be the only process on the machine using foreground printing mode. This is enforced by only allowing an application to perform foreground printing if it is running from the host **UNIX** PC. (Remote terminal applications cannot print in foreground mode.) The **iswind()** subroutine can be used to

determine if the program is running on the system's bit-mapped screen.

- Determine that the printer is not presently in use by the spooler. If it is, the application must not attempt to continue the foreground printing operation.

- Disable and lock out the spooler before printing.

- Reset, at the conclusion of foreground printing, the spooler enable/disable flag to the state it was in when the foreground printing operation was started.

The **UNIX** PC is configured to have a printer subsystem with two printers. One printer can be connected to the RS232-C port and another to the parallel port. See the *AT&T* **UNIX** *PC Owner's Manual* for a list of printers currently supported on the **UNIX** PC.

The first step in spooled printing is to determine on which printer you wish to print. If the application has simple, flat ASCII data to be printed, it can use the default printer and bypass this step [see lp(1) in the **UNIX** *System V User's Manual*]. Where the application cares about printer type, the **getpent** subroutine allows a program to get a list of all the configured, installed printers [see the getpent(3) in the **UNIX** *System V User's Manual*]. By making calls to **getpent**, you can tell which printers are available. When more than one acceptable printer exists, your program should ask the user which one to use.

The names returned by **getpent** are the same as the names in the Printer Setup menu in Administration, except that they are optionally suffixed by **_R** (for example, **ATT455_R**). Printers so suffixed are capable of accepting straight binary data with no interpretation by the driver or the spooler. Thus, if you are planning to print data with embedded printer control sequences, or others (such as graphics on a 470-class printer), you want the "raw" (**_R**) version of the printer. The nonraw

printers are subject to new-line conversion, for example. In any case, the printer type is obvious from the name, so you can customize the output for that particular printer.

Foreground Printing

In the foreground print mode, applications can bypass the spooler. This would be needed, for example, by a word processor printing on a single sheet of paper. In this mode, the word processor must know when printing has been completed and can prompt the user to change the paper. To be sure that no contention occurs between an application and the spooler, the application must first issue a disable command to the spooler.

Before disabling the printer, you must first make sure that no jobs are currently printing. The **getpent** subroutine will return this information. If a job is printing, you should not attempt to operate the printer in foreground mode and should issue an appropriate error message to the user.

Only if the printer is not in operation should foreground printing be done; then if you disable the printer, the spooler will not interrupt your printout until you are finished and have reenabled the printer.

The disable command is given by forking a process to execute the disable command [see enable(1) in the **UNIX** *System V User's Manual*]. This will stop the spooler if it is already running, and will prevent it from printing any other jobs while the application has use of the printer device.

To prevent a condition where multiple users on a single **UNIX** PC try foreground printing at the same time, foreground printing is only allowed for a user at the host **UNIX** PC. *This restriction must be enforced by the application program.* The program must issue an **iswind() TAM** call when the user wants to perform foreground printing. This **iswind()** call will return TRUE if the user is using the system's bit-mapped

console [see tam(3T) in the **UNIX** *System V User's Manual* for more details]. If the current user is not on the host **UNIX** PC, the application program must not allow the foreground printing operation to proceed.

Once all this is done, you must open the printer directly. **CENTRONICS®** interface printers can be opened as **/dev/lp** for "cooked mode" or **/dev/rawlp** for "raw mode". In cooked mode, various character mappings (for example, new-line mapping) are done for you—see lp(7) in the **UNIX** *System V User's Manual.* In raw mode, every bit goes out as is. This mode must be used when outputting graphics information or where the appplication does not want any character remapped by the operating system.

Serial printers are attached to the RS232-C port which is known as **/dev/tty000**. Serial printers can be opened as **/dev/tty000**. See termio(7) in the **UNIX** *System V User's Manual* for more information.

In many cases, the device to which the printer is attached can be inferred. In other cases, such as the AT&T 455 printer, this is not possible. When it is unclear what device the printer is attached to, applications should inspect the files **/usr/spool/lp/class/Serial** and **/usr/spool/lp/class/Parallel**.

The Spooler lp Command

The interface to the print spooler subsystem is through the **lp** command. The **lp** command can be issued by a user at the keyboard, or it can be *exec*'ed or **popen**'ed by a program that has data to print. File arguments or an input stream are expected; an optional argument naming the printer that should be used will be accepted. If no printer is specified, the default printer is used. Use the **UNIX** PC specific \-**a** option to give the request a name. That way, the user can identify the request later. This must be compiled to allow the User Agent

to identify the proper file when the user examines the printer queue.

Remote Console Support

In addition to a user on the host **UNIX** PC (with its bit-mapped screen and the integral keyboard and mouse), the **UNIX** PC can support remote terminals. For a list of terminals supported by the **UNIX** PC, see the *AT&T* **UNIX** *PC Remote Access User's Guide* located in the Communications Management binder.

All vendor-supplied applications must run on the **UNIX** PC's screen and keyboard. In addition, applications also should be capable of running on remote terminals. Application writers should thoroughly read the discussion of differences between TAM support for local and remote environments found in tam(3T) in the **UNIX** *System V User's Manual.*

Software Development Environment

The **UNIX** PC contains a complete C language development system, so software can be developed directly on the **UNIX** PC. While the C language is the "standard" language on the **UNIX** PC and is preferred for all uses, other languages, including those of several vendors, are supported. Among these are Assembler and versions of BASIC, COBOL, FORTRAN, and Pascal.

The **UNIX** PC Development Set, which is contained in the optional Utilities package, supplements the System Software Set to provide full **UNIX** System V for application development. A complete listing of the commands and application programs in the **UNIX** Utilities set is in Appendix C of the *AT&T* **UNIX** *PC Owner's Manual.* The development environment includes:

- **UNIX** System V with utilities
- **cu, uucp**, and **cpio** (part of the system software set)
- **vi** (and **ed** in system software)
- **cc, as**, and **ld**
- **make**
- **ar**
- the **sdb** debugger
- the **lp** print spooler (part of the foundation set)
- a full set of I/O libraries (**stdio, TAM**)
- **lint**
- **SCCS**

The **UNIX** PC supplies a global environment variable, $HOME, to point to the home directory for the current user. Application programs use this variable to locate any initialization files to be used for the current user.

User Agent

The standard **UNIX** PC user interface is the User Agent. The User Agent provides a window- and menu-based interface to the **UNIX** System V file system and to **UNIX** PC services. It allows the user to invoke, suspend, and resume applications in simultaneously present windows, and it provides access to peripheral devices. See Section 1 of this document for a description of the User Agent and the tools used to build it. The tools are available for use by any application; you are encouraged to use them.

One of the important features of the User Agent is the ability to "cut" and "paste" data from one application to another. The User Agent handles data transfer between applications on the Clipboard. Your application should support this feature.

While the User Agent is the standard interface, the standard **UNIX** System V shell is provided on the **UNIX** PC for use by "expert" **UNIX** system users as the job control interface to the operating system. Users who are interested in using the shell should see Chapter 9—**ACCESSING THE UNIX SYSTEM: EXPERT AND STANDARD MODES** in Section 1 of this guide.

Software Installation And Shell Interface To Applications

Any application program can be called in one of two ways:

1. From the **UNIX** System V shell, that is, "ordinary" **UNIX** operating system.

2. From the **UNIX** PC User Agent.

In either case, the application sees the same interface; on being started, it is passed any relevant parameters through the normal **UNIX argv** and **argc** calling arguments as defined in the C language. All applications must be able to utilize parameters passed in this way to provide for a clean interface to the User Agent.

Since the User Agent is the standard interface to the **UNIX** PC, all software must interface to it in a uniform way. The Administration features on the **UNIX** PC contain software to control the installation of software. By handling all installations through Administration, the user is presented with the same interface when installing new software, regardless of which vendor wrote the application being installed.

All vendors must use the supplied installation software.

The installation procedures for vendor software are detailed in Chapter 5—**SOFTWARE INSTALLATION.**

Remote Screen And Printer Option Settings

The **UNIX** PC system software has been written to make use of a variety of printers and terminals. To use the add-on peripherals, the user must have the option switches on the peripheral set to the values assumed by the **UNIX** PC software. The option settings required by the **UNIX** PC for proper operation of printers can be found in the *AT&T* **UNIX** *PC Owner's Manual.* The option settings required for proper operation of terminals can be found in the *AT&T* **UNIX** *PC Remote Access User's Guide* located in the Communications Management binder. You must insure that your code works with these option settings.

File System Integrity

Application programmers who have not worked previously with the **UNIX** operating system should be aware of a major difference between **UNIX** and single-tasking systems such as **MS-DOS.**

Part of the power of the **UNIX** system is derived from its disk caching scheme. **UNIX** maintains memory buffers of blocks going to the disk; actual writes to the disk are done only when the buffers are full. This means that even though an application has performed a disk write, the data can still be in memory and *not yet on the disk.*

To be sure that data is not lost, applications should follow these rules:

- Users should *never* be encouraged to turn the **UNIX** PC off without first going through the Shutdown procedure available in the Office and file system command menus.

- When an application wants to be sure a file is actually saved onto the disk, it should close the file and then issue a **sync()** call. The **sync** call will ensure that all memory buffers are flushed. Since a **sync()** takes time, the application should only use it when it is necessary to ensure that the data has indeed been written to disk.

Chapter 2

PROGRAMMING THE TELEPHONE PORT

This chapter describes how to access the **UNIX** PC's telephone port from a C language program. A general procedure for accessing the telephone port is outlined. Necessary system calls are briefly described for each step in the outlined telephone port access procedure.

Accessing the Telephone Port

The telephone port is typically accessed in the following order:

1. Open the telephone port's **UNIX** device.
2. Initialize the telephone port (includes storing the telephone port's original line and modem settings).
3. Manipulate the telephone port.
4. Return the telephone port to its original state.
5. Close the telephone port's **UNIX** device.

C programs that access the telephone port should follow the above sequence. A sample C program based on the above sequence can be found in Appendix B - **PROGRAMMING EXAMPLES.**

Necessary System Calls

Two sets of **UNIX** System V calls can be used to access the telephone port: **dial** and **undial**; and **ioctl**. Using **dial** and **undial** is the easiest way to access the telephone port. However, standard **ioctl** calls with phone port specific options provide more control over the telephone port than **dial** and **undial**.

The **dial** and **undial** system calls are preferred over **ioctl** calls for accessing the telephone port. While **ioctl** system calls provide more control over the telephone port, **dial** and **undial** make use of a lock file which helps manage access to the telephone port by competing, independent processes. The multitasking capabilities of the **UNIX** PC make the process management features of **dial** and **undial** very useful.

The following list describes the **UNIX** System V calls needed to program the **UNIX** PC's telephone port.

1. Open the telephone port.

   - file_descr = **open(/dev/ph1**, options) where the options are described in open(2) in the **UNIX** *System V User's Manual.*

2. Initalize the telephone port.

   - Save the original device setting.

     ioctl(file_descr, **TCGETA**, &orig_settings) where orig_settings is a **termio** data structure containing the original line settings [see termio(7) in the **UNIX** *System V User's Manual*].

   - Save the modem parameters.

     ioctl(file_descr, **PIOCGETP**, &orig_param) where

orig_param is an **updata** data structure containing the original modem parameter values.

- Assign the various input, output, line, and control flags.

  ioctl(file_descr, **TCSETA**, &new_settings) where new_settings is a **termio** data structure containing new line settings.

- Change the modem parameters to the desired values.

  ioctl(file_descr, **PIOCSETP**, &new_param) where new_param is a **updata** data structure containing the new modem parameter values.

3. Manipulate the telephone line.

   - **ioctl**(file_descr, pioc_option, &new_param) where pioc_option is one of the following:

     PIOCOFFHOOK—take the line off-hook

     PIOCDISC—disconnect the line

     PIOFLASH—flash the line

     PIOCHOLD—put the line on hold

     PIOCUNHOLD—reconnect the line

     PIOCLINESEL—select which line for offhook

     PIOCRECONN—reconnect line for more dialing

   - **ioctl(file_descr, PIOCDIAL,** &uudata) where uudata is a **uudata** data structure which holds the phone number. This call dials a single digit from the phone number. The last character dialed should be a "$".

4. Restore the original state of the phone port.

   - Restore the original phone port parameters.

     ioctl(file_descr, **PIOCSETA**, &orig_param)

   - Restore the original line settings.

     ioctl(file_descr, **TCSETA**, &orig_settings)

5. Close the **UNIX** device.

   close(file_descr)

For more information on programming the telephone port, see dial(3C) and ioctl(2) in the **UNIX** *System V User's Manual.*

Chapter 3

THE TERMINAL ACCESS METHOD

This chapter describes the Terminal Access Method (**TAM**) terminal interface support routines, which are provided with the optional **UNIX** Utilities of the **UNIX** PC, and reside in the **TAM** library. The **TAM** libraries must be linked to any applications that use the routines.

The routines provide an interface to the **UNIX** PC kernel windowing capabilities. While reading this section, programmers should have copies of tam(3T), window(7), form(3T), menu(3T), message(3T), paste(3T), and track(3T) in the **UNIX** *System V User's Manual* and the *AT&T* **UNIX** *PC* **UNIX** *System V Programmer's Manual.* These manual pages give detailed information about using the **TAM** facilities described below.

General TAM Functionality

TAM provides an access method for terminals. It allows access a variety of different terminals with a single consistent programming interface. When used to access the **UNIX** PC screen, keyboard, and mouse, **TAM** calls generally translate directly to system calls, so that processing overhead is minimized in this case.

When **TAM** is used to access a remote terminal, it simulates the **UNIX** PC kernel functionality, where possible. The interface model that the **UNIX** PC uses for terminal access is that of an ANSI X3.64 compatible terminal. When sending to a remote terminal, **TAM** translates the ANSI 3.64 escape sequences to the appropriate escape sequences for the particular terminal.

TAM uses the **termcap** data base to determine the capabilities of the remote terminal, and how they are accessed. In the remote terminal case, **TAM** simulates the kernel windowing capabilities for a particular application by maintaining a screen image, as well as those portions of windows that are overlayed.

In the remote terminal case, **TAM** optimizes screen access by using its knowledge of what is already on the screen. Note that this optimization is not available in the case of the screen on the **UNIX** PC itself because these calls are passed directly to the kernel. In either case, unnecessary screen I/O calls should be avoided. In particular, applications should try to avoid doing screen I/O one character at a time, and should instead, send complete lines or strings of text to the screen.

TAM Limitations

In the remote terminal case, only a single application owns the screen at any one time. This implies that multiple applications are not allowed to simultaneously access the screen of remote terminals.

Other differences for the remote terminal are: the mouse is not supported for the remote terminal (mouse-related subroutine calls are ignored), some sequences supported for the **UNIX** PC cannot be supported for remote terminals (the terminal can lack the capability), and fonts/graphics are not supported.

Special TAM Features

The **UNIX** PC supports multiple overlapping windows displaying simultaneously on the screen, and it supports the mouse. Detailed descriptions of these features are located in **TAM subroutine Calls** later in this chapter.

Windowing

Multiple overlapping windows can reside simultaneously on the **UNIX** PC screen. These windows can be owned by the same process (application) or by distinct processes. The covered portions of any windows are remembered on a pixel basis by the **UNIX** PC kernel.

The **UNIX** PC kernel maintains an ordering of the windows based on which window overlaps which other windows. The topmost window, in this ordering, is always completely visible. When a window is selected, it is brought to the top and can potentially overlap other windows or completely overlay them.

A window owned by a particular application can be manipulated without the consent of the application (e.g., if a user resizes the window from the User Agent). The characteristics of the window that can be changed are size, shape, position, depth, and other miscellaneous parameters. The application is notified of the change via a signal (SIGWIND). This signal is sent to the process group associated with the window.

For an application to respond intelligently to changes in its window, it must catch the signal (the default is that SIGWIND is ignored) and issue a **wgetstat()** call to determine how the window has changed. The **wgetstat()** call is described shortly.

An application also can change the characteristics of its own window. In this case, no signal is sent to the application. In fact, when any process in the process group associated with the window changes the characteristics of the window, no signal is sent.

Care should be taken when an application changes its own window, as this might be defeating the wishes of the user. For example, a very annoying process could be written that makes its window as large as the screen, and every time it receives the

SIGWIND signal, it resets its window to be on top and as large as the screen. Then the user is effectively blocked from running any other applications.

A single process can create multiple windows. In this case, it receives the SIGWIND signal when any of its windows are changed. The **wgetstat()** call can be used to determine which window was affected.

Windows have borders with optional icons to let the user manipulate what is displayed inside a window. Window borders come with the following border icons:

Icon	Key
Help	HELP
Cancel	CANCL
Scroll up	Shift-up arrow, Page up, Beg
Scroll down	Shift-down arrow, Page down, End
Scroll right	Shift-right arrow
Scroll left	Shift-left arrow
Move	
Resize	

The pressing of a mouse button while the Mouse pointer is on any of the border icons except Move and Resize is translated by the kernel into the corresponding keystroke. It is up to the application to respond to the keystroke. Move and Resize functions are handled by the kernel, not the application. Note that for the scroll up and scroll down icons, the key code returned depends on the mouse button pressed.

Each window is a separate device, and can have its **tty** modes set independently of the other windows. Window I/O is performed via file descriptors, so any open windows of a process are inherited by a child process. All of the usual **UNIX** system calls apply, so, for example, an application can set the "close on exec" bit via **fcntl()** to prevent access to

particular windows by a child process. See window(7) and termio(7) in the **UNIX** *System V User's Manual* for more information on the capabilities of the windows.

> ***Note:*** On a remote terminal, window id's are not file descriptors, and the **tty** modes of separate windows are not independently settable.

A process can also set up a child process to make its standard input and output point to a particular window. The usual technique is used, that is, close file descriptors 0, 1, and 2 (**stdin**, **stdout**, and **stderr**) and duplicate the window's file descriptor.

Window-changed signals are sent to the process group associated with the particular window. Thus, a parent process receives, by default, signals concerning windows created by its children. The **setpgrp()** and **wiocpgrp()** system calls can be used to prevent this. {For more details, see window(7) in the **UNIX** *System V User's Manual.*]

Windowing on a Remote Terminal

The major difference between the remote terminal and the integral **UNIX** PC screen and keyboard, from the application point of view, is the lack of kernel level window support. **TAM** supports windows transparently within a given process, but knows nothing about windows created by other processes.

In the remote terminal case, windows are accessed through **TAM**—provided window ID's (rather than through file descriptors), and child processes cannot inherit windows as previously described.

Each process has control over the entire screen, and windows created by other processes are erased when the new process issues a **winit()** call.

The **iswind()** call is used to determine if the terminal is remote or local.

Mouse Interface

Support of the mouse on the **UNIX** PC is provided by both high-level and low-level **TAM** routines. The low-level routines are documented in tam(3T) in the **UNIX** *System V User's Manual,* and provide for setting mouse parameters, getting mouse reports from the keyboard interface, parsing the mouse reports, and other functions. The high-level routines are documented in track(3T) in the **UNIX** *System V User's Manual.*

Of the two kinds of routines, **track()** routines provide the simpler interface to the mouse. The **track()** routines, in turn, use the low-level **TAM** mouse routines to achieve their purpose. The operation of the mouse and a detailed description of the reports returned by the mouse are given in the window(7) in the **UNIX** *System V User's Manual.*

Initially, mouse reports are disabled, so applications that do not use the mouse do not need to worry about them. When mouse reports are turned on by the application, they are returned to the application as a special 8-bit input code [or 7-bit input escape sequence, depending on the setting of the keypad(3) call] in the input stream.

In the default-enabled state, a mouse report is inserted in the input stream on each change of the button state (that is, each time any of the three buttons is pressed or released). Thus, these reports are buffered and are in sequence with any keyboard input. The **UNIX** PC does not recognize double-clicks or multibutton depressions.

The mouse report contains the mouse button state and the mouse cursor coordinates. This mouse report can be read and parsed by the application, or the application can use the **wreadmouse()** call, which reads the information from the input stream and returns it in a structure.

The programmer can enable the mouse to send a report when any of the following conditions occur:

- Any of the three mouse buttons change state
- The mouse cursor enters a predefined rectangle
- The mouse cursor leaves a predefined rectangle.

Any number of rectangles can be defined for mouse report purposes. You can also request any combination of the above three cases for the generation of mouse reports.

In the case of a remote terminal, mouse reports are never received, and the mouse-related subroutine calls are ignored.

> ***Note:*** Applications that desire to follow the mouse movement across the screen (for example, a graphics drawing package) should always use the track(3T) mouse motion routines to determine when the mouse has moved. This latter procedure will bog down the CPU and severely degrade performance for other users.

TAM Subroutine Calls

TAM is divided into high-level and low-level functions. Low-level **TAM** provides basic access to all terminal capabilities, while high-level **TAM** provides menu, forms, and mouse tracking capabilities. Except where noted, the **TAM** routines function on remote terminals as well as on the integral **UNIX** PC screen and keyboard.

TAM also includes a collection of subroutines that are compatible with the **curses** interface routines, so applications that currently use **curses** can be ported to the **UNIX** PC.

A description of all the **TAM** entry points follows.

Low-Level TAM

The following routines allow the application to create, modify and delete windows, and perform output to, and obtain keyboard and mouse input from, windows. The routines are grouped into four categories: setup, display interface, keyboard interface, and mouse interface. See tam(3T) in the **UNIX** *System V User's Manual* for more detail on these routines.

Setup

winit() Sets up the process for window access. It must be called before using any other window calls.

wexit() Should be called in place of **exit()**.

iswind() Boolean—Tells the user if he or she is in a hardware window environment, that is, the bit-mapped screen.

Display Interface

wcreate() Create a window.

wdelete Delete a window.

wselect() Makes the specified window the current window. If it is overlapped by any other window, it moves on top. A window is implicitly selected when it is created (**wcreate**) or modified (**wsetstat**).

wgetstat Gets the status of the window. The information returned includes the position and dimensions of the window, and whether it is the current window or not.

wsetstat Sets the status of the window. It can be used to change the size or shape of a window, and it selects the window implicitly. It also allows the application to turn the display of the window border on or off, and turn the display of border icons on or off.

The side-border and bottom-border icon displays are separately controllable. Border icons only display on the integral **UNIX** PC screen; they are used for accessing scrolling and paging functions using the mouse.

wputc() Outputs character to the window.

wputs() Outputs null-terminated string to the window.

wprintf() Does a **printf** to the window.

wslk() Outputs a null-terminated string to a single screen key, or optionally, outputs an entire 80-column line to write all screen keys at once.

wcmd() Outputs a null-terminated string to the command line.

wpromt() Outputs a null-terminated string to the prompt line.

wlabel() Outputs a null-terminated string to the window label line.

wrefresh() Flushes all output to the window. Output is normally buffered until input is read from the window. A special version of the **wrefresh()**

call is used in the remote terminal case to redisplay all windows known to the application.

wgoto() Moves the window's cursor to a specified row and column.

wgetpos() Gets the current position of the cursor in the specified window.

wprexec() On a remote terminal, Clears the screen and restores **tty** modes. On the **UNIX** PC, it creates a dimensionless window and duplicates the file descriptor to standard in, standard out, and standard error. It is used before exec of child process.

wpostwait() On a remote terminal, Restores **tty** modes and restores the process's windows. On the **UNIX** PC it is a no-op. It is used after a wait for child process to complete.

wgetsel() Returns the currently selected window. It is used after a window signal is received.

Keyboard Interface

wgetc() Gets a single character from **stdin**. It is the window equivalent of **getchar()**. The input stream from any keyboard is translated into **UNIX** PC keyboard equivalents (if **keypad()** is TRUE).

Mouse reports are also returned in the input stream. In the event of a mouse report, **wgetc()** first returns a unique code to indicate that a mouse report follows, then the report itself can be read character by character using **wgetc()**; or the **wreadmouse()** call can be used.

keypad() Determines how function keys are returned in a **wgetc()** call. (Function keys are in the left-hand and right-hand clusters on the **UNIX** PC keyboard.)

If flag=0, then this sets 7-bit mode—function keys return escape sequences for a **wgetc()** call. If flag=1, then this sets 8-bit mode—function keys return a single 8-bit character. If flag=2, then this sets nonmapped mode—function keys return the code generated by the terminal used.

Mouse Interface

wsetmouse Sets up parameters associated with the mouse. **wsetmouse()** also takes a pointer to a **umdata** structure which determines the report conditions for mouse motion and/or button state changes [see window(7) in the **UNIX** *System V User's Manual* for a discussion of the WIOCSETMOUSE and the **undata** structure for specific usage].

If no mouse is present or if the terminal is a remote terminal, **wsetmouse()** returns an error code.

wreadmouse() Gets the mouse state. The information returned consists of the mouse cursor coordinates and the mouse button state (whether each of the three buttons is up or down). Because the information is read from the input stream, **wreadmouse()** should be called only after a mouse code is returned by **wgetc()**.

High-Level TAM

menu() Creates a window large enough to hold the menu; it displays the menu, prompts the user to select menu item(s), and returns when a function key is pressed.

It displays single or multiple selection menus. The shape of the menu (number of rows and columns) can be set by the caller or determined automatically. A title should always be assigned to a menu to ensure proper window identification if the user accesses the Window Manager. A prompt can optionally be displayed with the menu.

The menu specification includes an array of named menu items. Each menu item can have an associated **int** value which is returned to the caller when that item is selected.

If the operator types in a string on the command line, that is matched against the menu items, and also returned to the caller.

form() Creates a window large enough to hold the form, displays the form, prompts the user to fill in the form, and returns when a function key is pressed. A title should always be assigned to a form to ensure proper form identification if the user accesses the Window Manager.

The form specification includes an array of named fields. Each field is defined to reside at a particular location (relative to the form), with a specified length.

An initial value for the field is passed in, and is returned with a modified value if the user has edited the field.

Optionally, a prompt can be displayed for each field and a menu of choices for the field can be specified. If specified, the menu is displayed only at user command.

curses-Compatible Calls

This is a description of the routines supported for **curses** compatibility. Combining these routines with native **TAM** calls is generally safe on output, but should be avoided on input.

The list is arranged by function, and each item includes a brief description of what the routine does. In describing the arguments, three special names are used.

1. The name **dumsc** represents a dummy screen variable that must be included. Technically, its name does not matter, but the name **stdscr** is suggested for compatibility with "standard" **curses**.

2. The name **dumflag** represents a dummy flag variable that should be included. Technically, its name does not matter, but the name TRUE is suggested for compatibility with "standard" **curses**.

3. The name **booflag** represents a Boolean flag with value TRUE or FALSE, indicating whether to enable or disable the option.

Initialization

These functions are called when initializing a program.

initscr() This initializes the **curses** data structures.

endwin This restores **tty** modes, moves the cursor to the lower left corner of the screen, resets the terminal to the proper nonvisual mode, and tears down all appropriate data structures.

Option Setting

These functions set options within **curses**. Initially, all of these options are disabled. It is not necessary to disable these options before calling **endwin**.

clearok(dumscr,dumflag) This is a dummy routine that is present for **curses** compatibility. No action is taken.

keypad(dumscr,booflag) This option enables the keypad of the user's terminal and sets the mode of translation of keys. See the description in *Keyboard Interface.*

leaveok(dumscr,booflag) Normally, the hardware cursor is left at the location of the virtual screen cursor. This option allows the cursor to be left wherever the update happens to leave it. It is useful for applications where the cursor is not used, since it reduces the need for cursor motions.

If possible, the cursor is made invisible when this option is enabled.

nodelay(dumscr,booflag) This option causes **getch** to be a nonblocking call. If no input is ready, **getch** returns **cmil**. If disabled, **getch** hangs until a key is pressed.

Terminal Mode Setting

These functions are used to set modes in a **TAM** emulation of the **tty** driver. The modes affect other **curses**-compatible calls, but not native **TAM** calls. The initial mode usually depends on the setting when the program was called. The initial modes documented here represent the normal situation.

cbreak()

nocbreak() These two functions put the terminal into and out of CBREAK mode. In this mode, characters typed by the user are immediately available to the program. When out of this mode, the tty driver buffers characters typed until **new-line** is typed. Interrupt and flow control characters are unaffected by this mode. *Initially, the terminal is not in CBREAK mode.* Most interactive programs using **curses** set this mode.

echo()

noecho() These functions control whether characters typed by the user are echoed as typed. Initially, characters are echoed as they are typed.

Authors of many interactive programs prefer to do their own echoing in a controlled area of the screen; or not to echo at all, so they disable echoing.

nl()

nonl() These functions control whether new-line is translated into carriage return and linefeed on output, and whether return is translated into new-line on input. *Initially, the translations do occur.* By disabling these translations, **curses** is able to make better use of the linefeed capability, resulting in faster cursor motion.

resetty()

savetty() These functions save and restore the state of the **tty** modes. **savetty** saves the current state in a buffer; **resetty** restores the state to what it was at the last call to **savetty**.

fixterm()

resetterm() These are two low-level curses routines used to change the **tty** modes between the two states: *normal* (the mode they were in before the program was started) and *program* (the mode needed by the program). **fixterm** puts the terminal into program mode, and **resetterm** puts the terminal into normal mode. These functions are useful for shell escapes and <Ctrl>-<z> suspensions.

Writing Text on the Virtual Screen

These routines are used to write text on the virtual screen. The upper left corner is always (0,0) not (1,1). The functions with the prefix **mv** imply a call to **move** before the call to the output function. Note that these routines do not necessarily cause characters to be output to the physical screen. For descriptions on how this occurs, refer to the section **Sending Output to the Terminal.**

Moving the Cursor

move(row,col) The cursor is moved to the given location. This moves the virtual cursor, not the physical cursor of the terminal.

Writing One Character

addch(ch)

mvaddch(row,col ch) The character **ch** is put on the virtual screen at the current cursor position. If **ch** is a tab, new-line, or backspace, the cursor is moved appropriately. If **ch** is a different control character, it is drawn in the ^**X** notation. The position of the cursor is advanced. At the right margin, an automatic new-line is performed. At the bottom of the scrolling region, if **scrollok** is enabled, the scrolling region will be scrolled up one line.

Writing a String

addstr(str)

mvaddstr(row,col,str) These functions write all the characters of the null-terminated character string **str** on the virtual screen. They are identical to a series of calls to **addch**.

Formatted Output

printw(fmt,args) This function corresponds to the standard I/O library function **printf**. The characters which would be output by **printf** are instead output on the virtual screen.

Clearing Areas of the Screen

clear() This function copies blanks to every position on the virtual screen and then calls **clearok**, arranging that the screen will actually be cleared on the next call to **refresh**.

clrtobot() All lines below the cursor are erased. Also, the current line to the right of the cursor is erased.

clrtoeol() The current line to the right of the cursor is erased.

erase() This function is equivalent to **clear()**.

Inserting and Deleting Text

delch() This function deletes the character at the current cursor position. All characters to the right on the same line are moved one position to the left.

deleteln() This function deletes the line that the cursor is currently on. All lines below are moved up one line, and the bottom line is cleared. The function returns the number of the deleted line.

insch(c) This function inserts the character **c** before the character at the current cursor position. The current character and all characters to the right on the same line are moved one position to the right.

insertln() This function inserts a blank line at the line that the cursor is currently on. The current line and all lines below are moved down one line, and the bottom line is lost. The function returns the number of the line deleted.

Sending Output to the Terminal

refresh() This function copies the contents of the virtual screen to the physical screen.

Input From the Terminal

getch() A character is read from the terminal. In **nodelay** mode, if there is no input waiting, the value **−1** is returned. In **delay** mode, the program hangs until the system passes text through to the program. Depending on the setting of **cbreak**, this will be after one character, or after the first new-line.

If **keypad** mode is enabled and a function key is pressed, the code for the function key is returned instead of the raw characters. If a character is received that could be the beginning of a function key (such as <Esc>), **curses** sets a one-second

timer. If the remainder of the sequence does not come in within one second, the character is passed through; otherwise, the function key value is returned. For this reason, there is a one-second delay after a user presses <Esc>, for example. (Use by a programmer of the escape key, <Esc>, for a single character function is discouraged.)

Reading Contents of the Screen

Reading characters from the screen is not supported.

getyx(dumscr,row,col) The current cursor position is placed in the integer variables **row** and **col**.

Video Attributes

attroff(at)

attron(at) These functions set the current attributes of the screen. The attributes can be any combination of A_STANDOUT, A_REVERSE, A_BOLD, A_DIM, A_UNDERLINE, A_STRIKE. These constants are defined in **TAM.h** and can be combined with the C | (or) operator.

The current attributes of the screen are applied to all characters that are written on the screen with **addch**. Attributes are a property of the character, and move with the character through any scrolling and insert/delete line/character operations.

attroff(at) This turns off the named attributes without affecting any other attributes.

attron(at) This turns on the named attributes without affecting any others.

Bells and Flashing Lights

beep() This sounds the audible alarm on the terminal.

flash() This calls the **beep()** routine.

Portability Functions

These functions have nothing to do with terminal-dependent character output, but can be needed by programs that use **curses**. Unfortunately, their implementation varies from one version of **UNIX** to another. They have been included here to enhance the portability of programs using **curses**.

baudrate()

baudrate These return the output speed of the terminal. The number returned is the integer baud rate (for example, **9600**) rather than a table index such as **B9600**.

flushinp()

flushinp These throw away any typeahead that has been typed by the user and has not yet been read by the program.

Chapter 4

PROGRAMMING THE *UNIX* PC

This chapter describes several **UNIX** System V program development tools. These tools are useful to programmers developing applications for the **UNIX** PC. The **UNIX** System V tools described in this chapter are (1) the C compiler, (2) libraries and shared libraries, (3) the Make program, and (4) loadable drivers. This chapter also lists a number of ways for improving application performance on the **UNIX** PC.

Calling the C Compiler

The **UNIX** PC C compiler is included with the optional **UNIX** utilities package. The **cc** command is the **UNIX** System V C compiler. The **cc** command is invoked by typing

cc [option] file_name

where file_name is a **UNIX** file containing a C language program. The file name should end with **.c**. By default, **cc** outputs the compiled program to the executable file **a.out**.

Example: To compile a C language program contained in the file **example.c**, type

cc example.c

The C compiler compiles **example.c** and outputs the object code file **a.out**.

Several options can be used with **cc**. Two frequently used options are **−c** and **−O**. The option **−c** causes **cc** to produce a linkable object file. For example, **cc −c example.c** produces the file example.o.

The option **−O** invokes an object code optimizer. The object code optimizer attempts to reduce the size of the object code.

For more information on calling the C compiler, see the cc(1) and ld(1) manual pages.

Libraries and Shared Libraries

A library is an executable file of C functions (e.g., printf). Frequently used functions can be stored in a library and then used by C programs that require the functions. Library routines called by a C program become part of the object code file produced when a C program is linked. For example, if a compiled C program is 10 K in size and the library functions called by the program are 25 K in size, the object code file produced by the link editor (ld) is 35 K in size.

The **UNIX** PC provides programmers with a shared library. A shared library is identical to a library except for the compilation of C programs using functions in a shared library. When a C program using shared library functions is compiled, a pointer to the shared library function, rather than a copy of the functions' object code, is included in the object code file.

The use of shared libraries in C language programs has the following advantages:

1. Memory demands on a system are reduced. Only one copy of a shared library function is needed regardless of the number of active processes using the function.

2. Disk usage is reduced. Programs using a shared library function do not carry the function's object code in their object code file.

3. Programs load faster since programs' object code files are smaller.

4. Programs run faster since programs' object code files are smaller.

5. Programs require less overlay swapping since the size of the program's object code files are reduced.

For more information on using shared libraries, see the shlib(4) manual pages.

The Make Program

Programs with only one or two modules can be compiled directly with **cc**. When systems grow to contain many modules or programs with many interdependencies, updating a module, finding dependent modules, and recompiling the affected modules is a significant task. The **make** command is a **UNIX** System V tool that maintains, updates, and regenerates groups of modules and programs.

The **make** program provides a method for maintaining up-to-date versions of programs that result from many operations on a number of files. The **make** program keeps track of the sequence of commands that create certain files and the list of files that require other files to be current before the operations can be done. Whenever a change is made in any part of a program, the **make** command creates the proper files simply, correctly, and with a minimum amount of effort. The **make** program also provides a simple macro substitution facility and the ability to encapsulate commands in a single file for convenient administration.

The basic operation of **make** is to update a target file by ensuring that all of the files on which the target file depends exist and are up-to-date. The target file is created if it has not been modified since the dependents were modified. The **make** program does a depth-first search of the graph of dependencies. The operation of the command depends on the ability to find the date and time that a file was last modified.

The **make** program operates using the following three sources of information:

- A user-supplied description file
- File names and "last-modified" times from the file system
- Built-in rules to bridge some of the gaps.

The **make** program is most useful for medium-sized programming projects. The **make** program does not solve the problems of maintaining multiple source versions of, or of describing, huge problems.

For more information on **make**, see Chapter 13—**A PROGRAM FOR MAINTAINING COMPUTER PROGRAMS-"make"** in the *AT&T* **UNIX** *PC* **UNIX** *System V Programmer's Guide.*

Loadable Drivers

Device drivers are required for any **UNIX** PC program that uses a device (e.g., terminal, keyboard, telephone port). **UNIX** System V allows hardware to be easily integrated because the **UNIX** System V architecture provides a uniform interface to every device. The role of a device driver is to translate the actions of the general interface provided by **UNIX** System V into actions understood by a particular device. Once a device driver is loaded, it has access to all kernel subroutines and global data. A loaded device driver is a part of the running kernel.

Loaded device drivers consume some main memory. Loadable device drivers can be loaded when their device is needed and removed after their device is no longer needed. This process frees some main memory for user processes. Loadable drivers should be used in applications software for devices that are not required frequently.

For more information on loadable drivers, see driver(7) and lddrv(1) in the **UNIX** *System V User's Manual.* For more information on how to write a loadable device driver for the **UNIX** PC you can write to:

AT&T Information Systems
Product Manager—**UNIX** PC Enhancements
Attn: Device Driver Information (Rm 6A45)
1776 On the Green
Morristown, NJ 07960

Improving Application Performance on the *UNIX* PC

The **UNIX** PC has exceptionally good benchmark performance. However, like any computer system, poor programming can result in slow performance. Programmers should strive for optimum performance when writing code. Application performance on the **UNIX** PC may be improved by the following steps:

1. Use short integer arithmetic operations whenever possible. Long integer and floating-point operations are considerably slower than short integer operations. For example, long integer multiplication is 10 times slower; floating point multiplication is 100 times slower.

2. Use the optimizer supplied with the compiler you are using. On the standard C compiler supplied with the **UNIX** System V utilities package, the "-o" option invokes the optimizer.

3. Profile the source code and identify those modules with long run-times. The program logic of these modules can be altered to avoid using the modules any more than possible. The modules can also be recoded into assembly language. Profiling facilities are available to program developers for the **UNIX** PC.

4. Use the **TAM** library for screen access.

5. Take advantage of the 2.5 Mb virtual memory of the **UNIX** PC. Application software ported from **MS-DOS** originals may not fully use the large virtual memory of the **UNIX** PC. Programs should be rewritten to take the best advantage of the **UNIX** PC's facilities.

6. Avoid extensive use of unique keys on the **UNIX** PC keyboard. Remote terminals may not be able to access features of an application if **UNIX** PC-specific keys are extensively used.

Chapter 5

SOFTWARE INSTALLATION

The **UNIX** PC offers an administration tool that allows users to install application software packages from within the menu structure of the Office. Provided here is an outline for the applications programmer of the installation procedure, and a detailed set of instructions for the creation of floppy disk sets that can be installed using the **UNIX** PC administration tools.

The applications programmer is assumed to have knowledge of **UNIX**, shell programming, and the structure of the **/usr/lib/ua** directory that enables the window manager to function.

To install software on the **UNIX** PC, the user is instructed to enter the Administration menu from the Office window, select Software Setup, and then select **Install Software From Floppy**. The User Agent prompts the user to insert the first floppy disk for the application software and then any others until all have been read.

At this point, the contents of the floppy disk have been read into a temporary area. An executable program supplied on the floppy disk is executed to move the software from the temporary area into the final locations, and to modify the system files so the software can be used by the customer.

Following is an outline of the procedure executed at installation time (Install.sh). The applications programmer is provided with enough details about the structure of the several required files to actually create an installable floppy set for the **UNIX** PC.

Also, some broad guidelines for the installation and removal programs are provided along with actual examples of the installation script that handles the installation of the AT&T **UNIX** PC Word Processor package.

The Approach

The transfer mechanism is **cpio**. The floppy disk will be created with a **cpio -ocB** call and read in with a **cpio -icBdua** call.

The key for the scheme to be user-friendly and to allow some error protection is to include several special files created by you, the applications programmer. These files MUST appear on the cpio list. The special files are:

- The *Size* file contains information that allows the **Install.sh** program to determine the total size of the floppy set. The precise format of this file is shown later. This file should be the **first** file on the floppy disk.

- The *Install* file is executable—it is executed AFTER the files are **cpio**'ed in from the floppy set and BEFORE they are moved from the temporary installation directory. The content of Install is entirely under your control. Much of what follows deals with the rules and tools you need to be aware of to set up the Install file.

- The *Name* file contains descriptive information regarding the application which can be made available to the user at installation time. The Name file format and content are described later.

- The *Remove* file is an executable file that is executed if the customer chooses to remove the software package. The simplest removal tool now is removal of all path names contained in the Files file.

This is a poor default case since it does not take into account any files used by more than a single application or which are part of the base system itself—work in this area will be required to ensure that your Remove script is safe and leaves the system in the same state it was in BEFORE your tool was installed. A later section deals with the specification of the Remove executable file.

- The *Files* file contains a list of relative path names that can be used by the Remove program if desired. It is required for reference even if not used by the Remove program. It should list each file contained on the floppy set separately with no wildcarding.

An Outline of the Installation Procedure

The shell script **Install.sh**, called by the Administration software, creates the floppy disk set containing the application tool. Its function is to read in the floppy set and execute the Install program supplied on the floppy set.

The installation procedure is available to all users by selecting the Administration object in the Office, selecting the Software Setup object, and then selecting the **Install Software From Floppy** object. On selection of this menu item, the executable shell script **Install.sh** in **/usr/bin** is run. The following steps outline this program from a user's point of view (this is documented in the *AT&T* **UNIX** *PC Installation Guide*):

1. The user is presented with an introductory window and asked to insert the first floppy disk and press the Return key.

2. The first operation is to determine if there is sufficient space on the file system to read in the entire floppy set. The procedure will read the **Size** file from the first floppy disk and make decisions based upon its content before proceeding. If there is insufficient space, the procedure is aborted. If no Size file exists, this takes the form of simply

asking the user how many floppy disks are in the application package and checking the maximum possible content against available space.

3. The entire floppy set is read in with a **cpio -icBdu** call after creating and moving to directory **/tmp/installed.** Note that this directory is created exclusively for the installation and is removed following completion of the operation. Note also that the path names on the **cpio** floppy MUST be relative (not begin with a /) so that other file systems are not affected at this stage.

4. If some specific files cannot be found, the user is advised that the floppy set cannot be installed using this procedure. The files required are ./Install, ./Name, ./Remove, and ./Size which are described in the following section, **Creation of the Floppy Set.**

5. The vendor-supplied program in **./Install** is now executed. It must do whatever is necessary to move the application out of **/tmp/installed** and into a more permanent file.

6. The **Install.sh** script now updates a few key files in **/usr/lib/ua** so that the menus needed for listing or removing installed software are kept up-to-date. The **CONTENTS** file in **/usr/installed** is also updated and the Remove script is renamed and located there. All this allows users to find out what applications they have installed and remove them if desired.

7. The directory **/tmp/installed** and whatever remains of its contents are now removed.

8. The user is told that the installation is complete.

Creation of the Floppy Set

The Format of the Floppy Disk

The output of your operations will be a floppy disk(s) which is created via the **cpio -ocB** command [see the cpio(1) manual page for specific meanings of the **ocB** options]. This is mandatory since the program that reads the floppy (**/usr/bin/Install.sh**) assumes this form.

The list of path names which drives the cpio has three restrictions:

- All path names MUST be relative (NOT begin with a /). This is because the files will first be **cpio**'ed into the **/tmp/install** directory before they are moved into their permanent locations by your Install procedure.
- The first entry on the list must be **Size**.
- The entries immediately following Size must be **Name**, **Remove**, and **Install**.

Following these special files should be the path names (without leading /) for all files to be placed on the floppy.

The floppy set should be clearly labeled with the name of the product (as it appears in the Name file), the version of the product (which can include the date), and the floppy number (for example, number 2 of 5) showing the total number in the package.

The Name File

The Name file is simply a file containing the name of the product as it is to appear during the installation and removal stages of the **UNIX** PC operation. It is a single line that can contain the product name as well as version information. Note that only the first 65 characters are displayed in the menus which are kept by the **Install.sh** and **Uninstall.sh** programs. The content of this file does not affect the Suffixes file or anything outside of the installation procedure, although consistency tends to be less confusing than random naming conventions.

The Size File

The contents of the Size file should be the number of blocks required by the application program package once it is installed in a one-line flat file. A block is defined as 512 bytes—the total size can be derived using the **du** command, if the files are situated correctly, or by calculating an approximate value based on the **ls -l** listings of the individual files which comprise the package.

Install.sh makes use of this file as follows:

- The file is extracted from the FIRST position on the floppy.
- The amount of free space in the system is computed (using **df** or the equivalent).
- If the number in the Size file is greater than the number of available blocks, the user is NOT allowed to install the package.

Until a possible future enhancement, the user is asked to provide a floppy count; total size is estimated based on 300,000 bytes per floppy.

When calculating the number to put in the Size file, it is generally a good idea to round up and to include any work area that can be needed during the Install procedure (for example, you delivered source code which is compiled during your Install execution—you would want to ensure space is sufficient to hold the **a.out** file). If you have a disk-size intensive application (for example, a data base or spreadsheet) you might want to require that more space be free to accommodate potential user files.

The actual number is up to you, but be sure not to let user's run their file system out of space while installing your tool. There is no protection beyond the Size file that prevents this.

What the Install Program Should Contain

The Install program contained on your floppy should do everything necessary to get the files that were read into **/tmp/install** fixed up and ready to use in the permanent file system. You can do anything that a shell script can do in the Install program. Look at the Install program for the Word Processor set (shown in Example 1). It is important to test your installation on a **UNIX** PC without your application to make sure it works the way you intended it to.

EXAMPLE 1: SAMPLE INSTALLATION SCRIPT

The Word Processor Install program

```
#
# first move all relevant files to their destination
#

echo ''Installing Word Processing files''

LIST1="wp wp_merge wp_print wp_rvw wpp_band wpp_diablo
      wpp_prtsh wpp_qume''
LIST2="wp.hip prtconfigfile''
LIST3="edit:W format"W''
```

```
for i in $LIST1
do
->mv $i /usr/bin
->chown bin /usr/bin/$i
->chgrp bin /usr/bin/$i
->chmod 755 /usr/bin/$i
done

for i in $LIST2
do
->mv $i /usr/lib/ua
->chown bin /usr/lib/ua/$i
->chgrp bin /usr/lib/ua/$i
->chmod 644 /usr/lib/ua/$i
done

for i in $LIST3
do
->mv $i /u/tutor/Filecabinet/practice
->chown tutor /u/tutor/Filecabinet/practice/$i
->chgrp users /u/tutor/Filecabinet/practice/$i
->chmod 644 /u/tutor/Filecabinet/practice/$i
done
#
# Now update the user agent special files
#

echo ''Updating special Office Manager files''

echo ''Name=Records'' > /tmp/t
echo ''Suffix=:R'' >> /tmp/t
echo ''Description=*Records'' >> /tmp/t
echo ''Default=Open'' >> /tmp/t
echo ''Open = EXEC -d /usr/bin/wp -n %n %o'' >> /tmp/t
echo ''Create = EXEC -d /usr/bin/wp -c -b -n %n %o'' >>
     /tmp/t
echo ''Help = EXEC -d /usr/bin/uahelp -h /usr/lib/ua/wp.hlp
     -t Records'' >> /tmp/t
echo ''Print = ERROR To print a Records file, open it & select
      Print'' >> /tmp/t

uaupd -r Records -a /tmp/t Suffixes

echo ''Name=Glossary'' > /tmp/t
echo ''Suffix=:G'' >> /tmp/t
echo ''Description=*Glossary'' >> /tmp/t
echo ''Default=Open'' >> /tmp/t
echo ''Open = EXEC -d /usr/bin/wp -n %n %o'' >> /tmp/t
echo ''Help= EXEC -d /usr/bin/uahelp -h /usr/lib/ua/wp.hip
```

```
      -t Glossary'' >>/tmp/t
echo ''Print=ERROR To print Glossary, open it & select Print''
     >> /tmp/t

uaupd -r Glossary -a /tmp/t Suffixes


echo ''Name=Document'' > /tmp/t
echo ''Suffix=:W'' >> /tmp/t
echo ''Description=*Document'' >> /tmp/t
echo ''Default=Open'' >> /tmp/t
echo ''Open = EXEC -d /usr/bin/wp -n %n %o'' >> /tmp/t
echo ''Create = EXEC -d /usr/bin/wp -c -n %n %o'' >> /tmp/t
echo ''Help= EXEC -d /usr/bin/uahelp -h /usr/lib/ua/wp.hlp
     -t Document'' >> /tmp/t
echo ''Print=ERROR To print Document, open it & select the
     Print'' >> /tmp/t

uaupd -r Document -a /tmp/t Suffixes
#
# Check if the user wants wp as his or her default editor
#

echo ''Do you want the word processor to be the default
     editor for''
echo ''normal text files? (Type y or n)''
ANSWER=`line`

if [ $ANSWER = ''y'' ]
then
->grep -v ''^EDIT'' /usr/lib/ua/Environment > /tmp/t
->echo ''EDIT=/usr/bin/wp'' >> /tmp/t
->cp /tmp/t /usr/lib/ua/Environment
fi

rm -f /tmp/t

echo ''Word Processor installation complete''
```

What the User Should See

The Install program runs as a shell program from within the Install window created when you choose the **Install Software from Floppy** choice in the **Software Setup** menu. Be aware of the window size you have to work with.

Your output goes to this window and you get input from this window (unless you make additional changes from within Install—like running C programs which you bring in off the floppy disk).

Getting the Programs Out of /tmp

When the Install program begins, you can assume that all the programs and other files on the floppy have been **cpio**'ed into **/tmp/install**. After modifying them as you wish (if at all) you are ready to move them into the permanent file system. Some issues to watch out for follow:

- *Permissions*: your best bet is to be sure they were correct when you created the **cpio**, and then use **ln** instead of **cp** or **mv**. Make sure the "executables" are executable and that read-write permissions are such that the normal user can use them as appropriate. Do not forget the **setuid** bit if appropriate.

- *Ownership*: do not supply files owned by local users on the machine which created them—be sure they are owned by **root** or **bin** in general.

- *Moving them*: do not use the **cp** command (you will double the space requirements needed during the install phase). The **/tmp/install** directory is eventually trashed by **Install.sh** after you are done with it, but the recommended procedure is **ln**.

If any intermediate files are created, use the **/tmp** or **/tmp/install** directory for them to ensure remnants are not left.

Where to Install the Application

Installation of applications directly in the Office is strongly discouraged. This clutters the Office and defeats the purpose for which the Office was intended. Normally, primary access to an application is through the Run, Open, and Create commands which is implemented by modifying the file **/usr/lib/ua/Suffixes**.

This method should not be used when an application does not create and operate upon special files. In these cases, applications should be installed in Services. Services is a special entry in the Office which provides access to services obtainable in no other way. To use Services, the application's Install script should first check to see if a Services entry already exists in the Office. If an entry does not exist, **/usr/lib/ua/Office** must be modified to create a Services entry and the file **/usr/lib/ua/Services** must be created. Services must then be modified appropriately. The application's Remove script should undo the above modifications, only removing Services from the Office if Services is now empty.

Modifying the Suffixes File

The **/usr/lib/ua/Suffixes** file should be updated if your application ever references or creates files unique to itself. The mechanism provides the ability to define a file suffix (like **.c**, **:W**, or **.bits**) and a set of rules which are to be applied if the file is ever used from within the Office and Filecabinet environment.

The ua(4) manual page provides a write-up on the Suffixes file as well as some other special files in **/usr/lib/ua**.

To modify these special files, a simple tool called **uaupd** is provided. It takes input from a provided file and appends it to these special files as appropriate. Documentation on **uaupd** has been provided in the uaupd(1) manual pages. The simplest

way to see how it works is to examine its use in the Word Processor Install (see Example 1).

Any suffixes you define should be limited to some non-alphanumeric character followed by a small number of alphanumeric characters. Check the **/usr/lib/ua/Suffixes** file you have on your machine for form, and to see some of the suffixes that have already been taken. You may wish to consult AT&T to reserve a suffix for your application.

> ***Note:*** Do **not** use a suffix reserved for another application. This will cause the application which originally reserved the suffix to not load properly.

Modifying the Menus

You have the ability to modify the contents of the window menus if appropriate. Check the ua(4) manual page on how to change them, and look at the Install script for the Word Processor (Example 1) to see how to use **uaupd** to change the Office menu.

Modifying the User Environment

The environment of your user is controlled through the use of Environment files in assorted directories. The global environment for the Office is controlled through **/usr/lib/ua/Environment** and the login specific environment is controlled by **/u/logname/Environment** where **/u/logname** is the standard home directory for a user called **logname**.

The needed environment variables can be set here; do not attempt to reset any system environment variables, but deal exclusively with environment variables unique to your application. Name the variables such that the likelihood of overlap with other applications is small.

Installing Libraries, Include Files, etc.

The biggest issue is *DO NOT OVERWRITE ANYTHING THAT IS ALREADY THERE.* Do NOT redefine any libraries or include files for the user.

Do NOT redefine any standard **UNIX** System V file or executable. In your procedures, you can supplement what is there now, but do not replace anything.

Notification to User

If any input is required from the user during the Install, use **stdin** and **stdout** (echo and read) to communicate. Do not use **UNIX** System V jargon and keep the user informed when activities take a while. Ask the user for verification along the way if necessary (use **message** if you want). Inform the user of any problems and offer solutions or allow for choices to be made. And, tell the user when it is all done.

Removal of Installed Software

Choosing the **Remove Installed Software** menu item from the **Software Setup** object within Administration starts the **Uninstall.sh** program. The **Uninstall.sh** program offers user's a list of the installed software and asks them to pick the one they want to remove. Once a valid choice is made, the Remove script that was originally supplied on the floppy disk in **/usr/installed** is used.

After finding it and executing it, the Installed software menus are adjusted to reflect the deletion. Note that the Remove script is run AS IS—how good it is and how safe it is depends on its author.

What the Remove Program Should Contain

Your Remove program, once executed, should leave the system in the same state it would have been in had your application never been installed. This is not easy; the install program should contain explicit path names of all files on the system which need to be removed (full path names). Recall that the Files file is not available when Remove is executed, and it might have contained different information anyway.

Any suffixes and menu items you put in must be removed. The situation might warrant your asking if the user wants to remove files in the system with your suffix on them. BE SURE NOT TO DO THIS UNLESS THE USER SAYS TO; the files belong to the user. Err on the side of safety when removing files, but do your best to be complete and be sure you delete files the user would see.

The user interface to the removal process is provided through the **/usr/bin/Uninstall.sh** program. A sample remove program as supplied on a floppy for the Word Processor set, and is shown in Example 2.

EXAMPLE 2: SAMPLE REMOVE SCRIPT

Word Processor Remove program

```
echo ''Removing Word Processing files''

LIST1="wp wp_merge wp_print wp_rvw wpp_band wpp_diablo
      wpp_prtsh wpp_qume''
LIST2="wp.hlp prtconfigfile''
LIST3="edit:W format:W''

for i in $LIST!
do
->rm -f /usr/bin/$i
done

for i in $LIST2
do
->rm -f /usr/lib/ua/$i
done

for i in $LIST3
do
->rm -f /u/tutor/Filecabinet/practice/$i
done

$ Now update the user agent special files

echo ''Updating special Office Manager files''

uaupd -r Records Suffixes
uaupd -r Glossary Suffixes
uaupd -r Document Suffixes

# Check if wp was the default editor

ED=`grep ''^EDIT'' /usr/lib/ua/Environment`
if [ $ED = ''EDIT=/usr/bin/wp'' ]
then
->grep -v ''^EDIT'' /usr/lib/ua/Environment > /tmp/t
->echo ''EDIT=/bin/ed'' >> /tmp/t
->cp /tmp/t /usr/lib/ua/Environment
->rm -f /tmp/t
fi
echo ''Word Processor removal complete''
```

APPENDIX A

UPLOADING AND DOWNLOADING FILES

PAGE

APPENDIX A

UPLOADING AND DOWNLOADING FILES

This appendix describes how files can be uploaded and downloaded between a **UNIX** PC and a host (mainframe or minicomputer) **UNIX** system.

System Requirements

To upload and download files, you will need the following:

- **UNIX** PC (with modem)
- Voice / Data Line (connected to the **UNIX** PC)
- Host **UNIX** system (with telephone ports)
- Host **UNIX** system account (login, password)

Connecting to the Host *UNIX* System

You must connect the **UNIX** PC to the host **UNIX** system in order to upload or download data. The easiest way to connect a **UNIX** PC to the host system is to use the **cu** command. **cu** calls up a **UNIX** system and manages an interactive conversation between two **UNIX** systems.

Before you use **cu**, check the **UNIX** PC's screen for the status of the Voice / Data lines. The status of the Voice / Data lines appears in the upper left corner of the **UNIX** PC's screen. At least one Voice / Data line must be set to Data. To change the status of a Voice / Data line, move the User Agent's highlight to Telephone in the Office. Press <Enter> or <B1> on the

mouse. Next, press <F3> (Line Select) to change a Voice / Data line. If you have two Voice / Data lines, you may need to press <F3> twice to change a line to Data. Line 1 or Line 2 should now be set to Data. Press <Exit>. See the *AT&T* **UNIX** *PC Telephone Manager User's Guide* for more information on the setup and selection of Voice / Data lines.

To use **cu**, you must first enter the *UNIX System* from the Office of the **UNIX** PC. You can do this by moving the User Agent's highlight to *UNIX System* in the Office and pressing <Enter>, or <B1> on the mouse. Once you have entered the **UNIX** PC's **UNIX System**, type:

cu -l /dev/"telephone port" "Host system's telephone number"

where telephone port can be ph0 (if your data line is the **UNIX** PC's Line 1) or ph1 (if your data line is the **UNIX** PC's Line 2). The **cu** command dials the host **UNIX** system's telephone number and connects the **UNIX** PC to the host system. The host system's login message should then appear. After you have correctly entered a login and password, you can upload and download files.

Uploading Files

Uploading a file transfers a copy of a file on a remote terminal (in this case, a **UNIX** PC) to a host system (in this case, a minicomputer or mainframe running **UNIX**). To upload a file from a **UNIX** PC to a host **UNIX** system:

1. Connect the **UNIX** PC to the the host **UNIX** system (see the above section).

2. Type ~%put "file name (**UNIX** PC)" "file name (host **UNIX** system)".

3. Wait until the shell prompt, $, appears.

The transferred file can now be accessed by the host **UNIX** system. The transferred file will appear under the second file name in the ~% put command.

Downloading Files

Downloading a file transfers a copy of a file on a host system (in this case, a **UNIX** system) to a remote terminal (in this case, a **UNIX** PC). To download a file from a host **UNIX** system to a **UNIX** PC:

1. Connect the **UNIX** PC to the host **UNIX** system (see **Connecting to the Host UNIX System**).

2. Type ~% take "file name (host **UNIX** system)" "file name (**UNIX** PC)".

3. Wait until the shell prompt, $, appears.

The transferred file can now be directly accessed through the **UNIX** PC's **UNIX** system or indirectly accessed through the Filecabinet. The transferred file appears on the **UNIX** PC under the second file name in the ~%**take** command.

For more information on transferring data between the **UNIX** PC and host **UNIX** systems, see the cu(1C) and phone(7) manual pages.

APPENDIX B

PROGRAMMING EXAMPLES

PAGE

APPENDIX B

PROGRAMMING EXAMPLES

This appendix contains several **UNIX** PC programming examples. The programs demonstrate how to use the **TAM** libraries to create windows, alternate character fonts, and use the mouse. An example of programming the **UNIX** PC telephone port is also included.

Program Example 1: Using TAM to Create and Manipulate Windows—Using Makefiles

```
/*************************************************************************/
/*                                                                       */
/* Name:        wind.c                                                   */
/* Compiler:    cc                                                       */
/* Machine:     AT&T UNIX PC                                             */
/* Description: To demonstrate how to use the tam libraries and windowing */
/*              software on the UNIX PC.                                 */
/* Author:      Elliot Rappaport                                         */
/*                                                                       */
/*************************************************************************/

#include <tam.h>
#include <kcodes.h>
#include <message.h>
#include <stdio.h>


short wn;
struct  wstat curstat;                  /* Used to change current window. */
main()
{
        register int c;

        /*
          Before any window operations can be performed several routines
          must be called.  The winit() call sets up the process for window
          access.  The keypad() routine indicates that we want function
          keys to return a single 8-bit character for each key stroke.  And
          the wcreate() routine creates the window based on the supplied
          arguments.  Since TAM reconfigures the existing window for the first
          wcreate() call, we use clear() to make sure the window is empty.
          This feature is more apparent when applications are executed from
```

```
  the shell directly.                                                    */

winit();
keypad(0,1);

wn = wcreate(1, 0, 16, 50, (BORDVSCROLL | BORDHELP | BORDCANCEL));
clear();

wprintf(wn, "This is window number = %d\n\n", wn);
wprintf(wn, "Use the mouse to click on one of the icons or\n");
wprintf(wn, "the keyboard to enter data.  Hit EXIT or the\n");
wprintf(wn, "Cancel Patch to exit.\n\n");
wprintf(wn, "I know about the following keys:\n");
wprintf(wn, "UP, DOWN, ENTER, HELP, CANCEL, and EXIT\n");

for (;;) {
        c = wgetc(wn);

        switch(c) {

        case    Up:
        case    RollUp:
                        domessage("UP KEY");
                        break;

        case    Down:
        case    RollDn:
                        domessage("DOWN KEY");
                        break;

        case    Enter:
                        domessage("ENTER KEY");
                        break;

        case    Help:
                        domessage("HELP KEY");
                        break;

        case    Cancl:
        case    s_Cancl:
        case    Exit:
        case    s_Exit:
                        domessage("EXIT or CANCEL KEY");
                        goto bailout;
                        break;

        default:
                        domessage("Unknown KEY");
                        break;

        }
}
```

```
        /*
          Restore to 24 X 80 borderless window before we leave.
          The clear() routine makes sure we have an empty window and
          wexit() must be called to reset the parameters set by winit()
          (e.g., tty modes).
                                                                    */

bailout:
        clear();
        curstat.begy = 1;
        curstat.begx = 0;
        curstat.height = 24;
        curstat.width = 80;
        curstat.uflags = NBORDER;

        wsetstat(wn, &curstat);
        wexit(0);
}


domessage(s)
char *s;
{
        message(MT_INFO, NULL, NULL, s);
}
```

APPENDIX B

Program Example 2: Using TAM to Print and Alternate Character Fonts

```
/*************************************************************/
/*                                                           */
/*  chartest  -  a program to print the character fonts on   */
/*               the PC-7300                                 */
/*                                                           */
/*  Author:  Kevin Redden                                    */
/*           AT&T-IS                                         */
/*                                                           */
/*  Last Modified: 11/29/84                                  */
/*                                                           */
/*  This program will load several alternate character fonts */
/*  and will then use the two different methods available    */
/*  to switch between them. If only two fonts are to be      */
/*  used, the ascii "shift out" and "shift in" characters    */
/*  can be used to alternate between the sets. If more fonts */
/*  are to be used (up to 8 fonts per window), then the fonts */
/*  must be explicitly selected.                             */
/*                                                           */
/*                                                           */
/*************************************************************/


/* load the include files needed by this program        */
#include <tam.h>
#include <sys/window.h>
#include <sys/font.h>

main()
{

        short wnl;      /* window file descriptor       */
        short i;
        short er;       /* error return var.    */

        /* set up the structure for the character fonts */
        struct ufdata ufd;


        winit();        /* init the window system       */
        /* create a borderless window                   */
        wnl = wcreate(1,0,24,80,NBORDER);

        /* clear the screen     */
        clear();



        /*********************************************/
```

```
/*************************************************/
/*                                               */
/*  The first test will load two fonts, then     */
/*  switch between the two with the ascii        */
/*  shift in/out characters. This only works     */
/*  for a maximum of two fonts                   */
/*                                               */
/*************************************************/
/*************************************************/

/* write with the standard font */
wputs(wn1, "This is the standard Font.\n");

/* print the character set      */
for (i=0x20; i<110; i++)
{
        wprintf(wn1,"%c",i);
}
wprintf(wn1, "\n\n");

/*************************************************/
/* the fonts are stored in /usr/lib/wfont        */
/* some of the fonts supported are:              */
/*      monitor.8.ft    mosaic.8.ft              */
/*      special.8.ft    system.8.ft              */
/*      system.r.8.ft                            */
/*************************************************/

/*********************************************************/
/* now load the 2nd font slot (slot 0 is reserved)       */
/*                                                       */
ufd.uf_slot = 1;        /* specify the slot number       */
strcpy(ufd.uf_name, "/usr/lib/wfont/special.8.ft");
er =1;  /* set no error */
er = ioctl(wn1,WIOCLFONT, &ufd);
if (er != 0)
{
        printf("1st ioctl error %d\n",er);
}

wputs(wn1, "This is the alternate font.\n");

/*  an ascii 'shift out' (016) will shift to 1           */
/*  alternate font                                       */
wputc(wn1, '\016');

/* print the character set      */
for (i=0x20; i<110; i++)
{
        wprintf(wn1,"%c",i);
}
wprintf(wn1, "\n");
```

```
/*                                                          */
/*  now shift back to the system font                       */
/*                                                          */
/* an ascii 'shift in' (017) will shift to the              */
/* main font                                                */
wputc(wn1, '\017');
wputs(wn1, "We are now back in the main font.\n");



/*********************************************************/
/*********************************************************/
/*                                                          */
/* Now we will manipulate multiple fonts, switching         */
/*  explicitly between them (as opposed to using            */
/*  the shift in/out method).                               */
/*                                                          */
/*  The fonts used will be:                                 */
/*       slot 0 = system font                               */
/*       slot 1 = special.8.ft                              */
/*       slot 2 = mosaic.8.ft                               */
/*       slot 3 = monitor.8.ft                              */
/*                                                          */
/*********************************************************/
/*********************************************************/



wputs(wn1, "\n\n\nNow run the multiple font test.\n");

/*********************************************************/
/*  load the 3rd font slot                                  */
/*                                                          */
ufd.uf_slot = 2;
strcpy(ufd.uf_name, "/usr/lib/wfont/mosaic.8.ft");
er =1;  /* set no error */
er = ioctl(wn1,WIOCLFONT, &ufd);
if (er != 0)
{
        printf("2nd ioctl error %d\n",er);
}

/*********************************************************/
/*  load the 4th font slot                                  */
/*                                                          */
ufd.uf_slot = 3;
strcpy(ufd.uf_name, "/usr/lib/wfont/monitor.8.ft");
er =1;  /* set no error */
er = ioctl(wn1,WIOCLFONT, &ufd);
if (er != 0)
{
```

```
        printf("3rd ioctl error %d\n",er);
}
/* print this in the system font        */
wputs(wn1, "This is the slot 1 font\n");

/*************************************/
/* shift to slot one specifically       */
wputs(wn1, "\033[11m");


/* print the character set      */
for (i=0x20; i<110; i++)
{
        wprintf(wn1,"%c",i);
}

/* line feed and print message  */
write (wn1, "\033[10m", 5); /* switch to sys font       */
wprintf(wn1, "\n\n");
wputs(wn1, "this is the slot two font\n");

/*************************************/
/* shift to slot two specifically       */
wputs(wn1, "\033[12m");

/* print the character set      */
for (i=0x20; i<100; i++)
{
        wprintf(wn1,"%c",i);
}
write (wn1, "\033[10m", 5); /* switch to sys font       */
wprintf(wn1, "\n\n");
wputs(wn1, "this is the slot three font\n");

/*************************************/
/* shift to slot three specifically     */
wputs(wn1, "\033[13m");

/* print the character set      */
for (i=0x20; i<100; i++)
{
        wprintf(wn1,"%c",i);
}
wprintf(wn1, "\n\n");

/*************************************/
/* shift back to the system font        */
wputs(wn1, "\033[10m");
wputs(wn1, "We are now back in the main font.\n\n");


/***********************/
```

```
        /* unload the fonts      */
        ioctl(wnl, WIOCUFONT, &ufd);     /* unload slot 4        */
        ufd.uf_slot = 2;
        ioctl(wnl, WIOCUFONT, &ufd);     /* unload slot 3        */
        ufd.uf_slot = 1;
        ioctl(wnl, WIOCUFONT, &ufd);     /* unload slot 2        */

        wdelete(wnl);
        wexit();
}
```

Program Example 3: Using TAM to Manipulate the Mouse and Windows

```
/*      bounce
 *      coffin 4/6/85
 *
 *      This is the "pong" game....
 */

#include <stdio.h>
#include <fcntl.h>
#include <tam.h>
#include <termio.h>
#include <menu.h>
#include <sys/font.h>
#define WINX 50
#define WINY 18
#define SPEED 10
#define MX (WINX*9)-32
#define MY (WINY*12)-18

/* this structure holds the current mouse state....  */
struct curmouse {
        int xp, yp, but, reason;
} xx;

struct umdata mouse;
short wn, ws;
struct icon myicon;

int fin();
double drand48();
char *name, *getenv();
void setup();

unsigned short ball[32];
short myscore, youscore;
int beepflag, xinc, yinc, xball, yball, diff=3;

main () {
        struct termio ttt;

        setup();
        start();

        /*  this is the main loop.... */
        for(;;) {
                domouse();
                wrastop( wn, ball,4,0,0,0,0,(unsigned short)xball,
                        (unsigned short)yball,32,16,SRCSRC,DSTCAM,0);
```

```
                xball += xinc;
                yball += yinc;

                /* if ball is at top, make increment positive  */
                if( yball <= 4 ) {
                        if( beepflag ) beep();
                        yinc = abs(yinc);
                }
                /* if ball is at bottom, make increment negative  */
                else if( yball >= MY ) {
                        if( beepflag ) beep();
                        yinc = -abs(yinc);
                }
                /* if ball is at left side, player may have a point */
                if( xball <= 5 ) {
                        if( beepflag ) beep();
                        if( yball > WINY*6-(diff*12)-12 &&
                            yball < WINY*6+(diff*12)-6) {
                                ++youscore;
                                score();
                        }
                        else xinc = abs(xinc);
                }
                /* if ball is at right side, you lose.... */
                else if ( xball > MX ) {
                        if( beepflag ) beep();
                        ++myscore;
                        score();
                }

                /* if ball is near the paddle.... */
                else if( xball >= (xx.xp-abs(xinc)-10) &&
                         xball <= (xx.xp-10) &&
                         yball>=(xx.yp-18) &&
                         yball<=(xx.yp+8)) {
                        if( beepflag ) beep();
                        yinc = (yball-(xx.yp)+8)/2;
                        xinc = -xinc;
                }

                wrastop( wn, ball,4,0,0,0,0,(unsigned short)xball,
                        (unsigned short)yball,32,16,SRCSRC,DSTOR,0);
        }
}

fin() {
        flushinp();
        clear();
        wprintf( wn, "\033[=0C" );
        wprintf( ws, "\033[=0C" );
        wdelete( wn );
        wdelete( ws );
        wexit( 0 );
```

```
}

/*      this one will parse the mouse report  */
int domouse() {
        int key, i;

        wsetmouse( wn, &mouse );
        key=wgetc( wn );
        switch( key ) {
                case '\177':
                case Exit:
                case Cancl:
                case s_Cancl:
                        fin();
                        break;
                case Mouse:
                        wreadmouse( wn, &xx.xp, &xx.yp, &xx.but, &xx.reason );
                        break;
                default:
                        return( (-1 ) );
                        break;
        }

        if( xx.but&4 && xx.reason&MSDOWN ) beepflag=(beepflag==0)?1:0;
        if( xx.but&1 && xx.reason&MSDOWN ) {
                if( diff <= 1 ) diff = 3;
                else --diff;
                for(i=WINY/2-3; i<WINY/2+3; ++i ) mvaddch(i,1,' ');
                for(i=WINY/2-diff; i<WINY/2+diff; ++i ) mvaddch(i,1,'\177');
        }

        return( 0 );
}


/*  this one keeps score, and does setup between points */
score() {
        int i;

        wprintf( ws, "\n\n  PC7300: %d\n  %s: %d", myscore, name, youscore );
        if( myscore >= 15 ) {
                newgame( 0 );
                return( 0 );
        }
        else if( youscore >= 15 ) {
                newgame( 1 );
                return( 0 );
        }
        else if( youscore > 7 && youscore < 12 ) xinc = SPEED + SPEED/2;
        else if( youscore > 11 ) xinc = SPEED * 2;
        else xinc = SPEED;
        xball = 24;
        yball = (int)(drand48()*(double)(WINY-3)*12.0)+18;
```

```
        for(i=WINY/2-diff; i<WINY/2+diff; ++i ) mvaddch(i,1,'\177');
        refresh();
        sleep( 2 );
}


/* this one does the initialization for a new game */
start() {
        int i;
        static int games = 0;

        myscore = youscore = 0;
        xinc = SPEED;
        yinc = 3;
        xball = 24;
        yball = (int)(drand48()*(double)(WINY-3)*12.0)+18;

        wselect( wn );
        wprintf( ws, "\n\n" );

        sleep( 3 );
}

/*  this one displays a menu for a new game */
int newgame( who )
        int who;
{
        menu_t mnu;
        static mitem_t cc[] = {
                "Yes", 0, 0,
                "No", 0, 1,
                "Don't Know", 0, 2,
                0, 0, 0
        };

        if( who )  mnu.m_title = "Good game!  Another?   ";
        else mnu.m_title = "You lose!  Try again?";
        mnu.m_label = "Bounce.... by S.Coffin";
        mnu.m_flags = M_SINGLE ;
        mnu.m_curi = cc;
        mnu.m_cols = 1;
        mnu.m_items = cc;

        menu( &mnu, M_DESEL | M_BEGIN | M_INPUT | M_END );

        if( mnu.m_selcnt != 1 || cc[1].mi_flags == M_MARKED ) fin();
        start();
        return( 1 );
}

/*      this is used on first initialization   */
void setup() {
        int i, w_id;
```

```
/* start with a clean window  */
close( 0 );
close( 1 );
close( 2 );
w_id = open( "/dev/window", O_RDWR );
dup( 0 );
dup( 0 );
winit();
if( !iswind() ) {
   fprintf(stderr,"\nSorry, you must use bit-mapped display!\n\n");
   wexit( (-1) );
}

beepflag = 1;
name = getenv( "LOGNAME" );

/* set up the ball */
ball[4] = ball[6] = ball[24] = ball[26] = 0xf000;
ball[5] = ball[7] = ball[25] = ball[27] = 0x000f;
ball[8] = ball[10] = ball[20] = ball[22] = 0xfe00;
ball[9] = ball[11] = ball[21] = ball[23] = 0x007f;
ball[12] = ball[14] = ball[16] = ball[18] = 0xff00;
ball[13] = ball[15] = ball[17] = ball[19] = 0x00ff;

/* set up my paddle (mouse icon) */
myicon.ic_fc.fc_hs = 32;
myicon.ic_fc.fc_vs = 20;
myicon.ic_fc.fc_ha = -32;
myicon.ic_fc.fc_va = -10;
myicon.ic_fc.fc_hi = 0;
myicon.ic_fc.fc_vi = 0;
for( i=1; i<64; i+=2 ) myicon.ic_raster[i] = 0xf000;

/*  ok, open a window now.... */
ws = wcreate (6, 56, 2, 18, 0 );
clear();
wlabel( ws, "Scoreboard" );
wuser( ws, "Scoreboard" );

wn = wcreate (2, 2, WINY, WINX, BORDCANCEL );
clear();
wlabel( wn, "Bounce....                          v1.4" );
wprompt( wn, "Click left button to toggle sound,\
right button to change difficulty." );
wcmd( wn, "For best performance, close all other\
applications and windows." );
wuser( wn, "Bounce" );

keypad (0, 1);

/* turn off cursor */
wprintf( ws, "\033[=1C" );
wprintf( wn, "\033[=1C" );
```

```
for(i=WINY/2-diff; i<WINY/2+diff; ++i ) mvaddch( i, 1, '\177' );

/*  establish mouse control  */
mouse.um_flags = MSOUT | MSDOWN | MSUP;
mouse.um_x = 0;
mouse.um_y = 0;
mouse.um_w = 1;
mouse.um_h = 1;
mouse.um_icon = &myicon;
wsetmouse( wn, &mouse );
```

Program Example 4: Manipulating the Telephone Port

```
/*      call.c
 *
 *      J. A. Kutsch & S. Coffin        AT&T-IS
 *      July 17, 1985
 *
 *      This is a simple program to make voice calls from the
 *      shell on the AT&T UNIX PC 7300
 */

#define LINE "/dev/ph0"          /* use /dev/ph1 if you want to, but it  */
                                 /* should be configured for voice       */

#include <sys/types.h>
#include <signal.h>
#include <fcntl.h>
#include <sys/phone.h>
#include <stdio.h>

struct  updata mydata;
struct  uddata dialdata;
int fd;

main( argc,argv )
        int argc;
        char *argv[];
{
        char in[80], *s;
        void giveup();

        /* get number */
        if( argc > 1 ) s = argv[1];
        else {
                printf( "Give number to dial:  " );
                gets( in );
                s = in;
        }

        /* open phone device */
        fd = open( LINE, O_RDONLY );
        if( fd <= 0 ) {
                fprintf( stderr, "%s: Can't open %s\n", argv[0], LINE );
                exit( 1 );
        }

        /* set signal catchers */
        signal( SIGQUIT, giveup );
        signal( SIGINT, giveup );
        signal( SIGHUP, giveup );
```

```
        /* set phone parms */
        mydata.c_lineparam = VOICE | DTMF;
        mydata.c_waitdialtone = 5;
        mydata.c_waitflash = 500;
        mydata.c_feedback = SPEAKERON | NORMSPK;

        /* go off hook to make call */
        ioctl( fd, PIOCOFFHOOK, &mydata );

        /* wait for dial tone */
        for(;;) {
                ioctl( fd, PIOCGETP, &mydata );
                if( mydata.c_linestatus & DIALTONE ) break;
        }

        /* dial the number */
        while( *s ) {
                dialdata.dd_digit = *s++;
                ioctl( fd, PIOCDIAL, &dialdata );
        }
        dialdata.dd_digit = '$';
        ioctl( fd, PIOCDIAL, &dialdata );

        /* wait for set to go off hook */
        for(;;) {
                ioctl( fd, PIOCGETP, &mydata );
                if( mydata.c_linestatus & SETOFFHOOK ) break;
        }

        /* connect phone */
        ioctl( fd, PIOCGETP, &mydata );
        mydata.c_feedback &= ~(SPEAKERON);
        ioctl( fd, PIOCSETP, &mydata );

        close( fd );
        exit( 0 );
}

/* catch signals and exit   */
void giveup() {
        ioctl( fd, PIOCDISC, &mydata );
        close( fd );
        exit( 1 );
}
```

999-801-314IS