DATAPOINT

RMS

Interactive Software

Design Practices

61695

Preface

This document presents a set of interactive software design standards that apply to all application software developed to run under the RMS™ operating system, version 2 or later. The standards have been developed in consultation with software designers and architects, system programmers, development managers, customer support representatives, product marketing personnel, and corporate management.

This document is intended for system programmers writing software for DATAPOINT®. It assumes a working knowledge of the RMS operating system and the DATAPOINT software development environment.

NOTES

CONTENTS

NOTES

CHAPTER 1. INTERACTIVE SOFTWARE AT DATAPOINT

Contents

The Interactive Model

DATAPOINT has determined there is a need to design and implement software that conforms to a set of uniform standards. It has now developed a generally applicable model for interactive software behavior and is adopting this interactive model for future RMS software.

A new keyboard has been designed to support the model through certain dedicated keys and internal processing. The command interpreter of the RMS operating system has been modified to support the model. It is now time to define the interactive model formally so that RMS user software can be designed and implemented in accord with its tenets.

This document describes the interactive model and presents specific standards that will ensure its successful implementation. You should use it at first to learn about the interactive model. Later it will serve as a reference manual containing the answers to specific questions you encounter in designing and implementing RMS user software.

Four Interactive Techniques

The interactive model consists of four distinct techniques:

- **The Miller Column Technique:** The Miller column technique allows programs to represent hierarchically structured information in a graphically interactive context. It is ideally suited to programs that deal primarily with tree-structured data.
- **The Ring Menu Technique:** The ring menu technique is the standard method for allowing users to specify commands and options relating to

the execution of a program. It integrates effectively into Miller column, field keyin, and freeform text editing applications.

- **The Field Keyin Technique:** Field keyin is almost as old as data processing itself. However, the interactive model defines some specific behavior that all progams invovled with field keyin must support.
- **The Freeform Text Editing Technique:** Freeform text editing enables programs to collect arbitrary streams of characters from a user. As with field keyin, the interactive software model defines some specific behavior that all programs involved with freeform text editing must support.

These four techniques provide sufficient interactive opportunities for all RMS-targeted software. This document defines the keystroke processing and screen animation that support each of these techniques and clearly identifies opportunities for application-specific extensions.

Benefits of Software Standards

The interactive software standards are part of an overall effort to change the image that DATAPOINT software conveys to the RMS user community. They will provide tangible benefits to

- **Users,** who may have been confused when they migrate between DATAPOINT software environments and discover they have little or nothing in common;
- **Programmers,** who have previously had insufficient information about the direction in which DATAPOINT wants its software development efforts to move;
- **Development managers,** who can use the standards in product design, business planning, and evaluating programmers' efforts; and
- **Certification personnel,** who can evaluate products based on their compliance with established standards.

Users: Our primary objective is to place users in an environment which encourages the natural mechanisms of intuition and heuristic learning. It is the responsibility of each interactive program to achieve this objective. However, we cannot succeed throughout our product line unless each program is consistent not only within itself, but also with the entire set of products that makes up a user's interactive context. The software standards, then, are the key to ensuring that we achieve our primary objective.

After using just one program, users will understand the basic rules for using any DATAPOINT software product. They will not have to learn to cope with a separate interactive environment for each type of problem they use our software to solve.

Because basic operating procedures remain constant, more sophisticated users will be able to learn advanced techniques more easily. Even naive users will develop greater interactive capabilities because they are not constantly relearning the basics.

Software Developers: DATAPOINT programmers and development managers will benefit too. The interactive model and the standards contained in this document provide measurable criteria by which to evaluate new proejcts. You will be able to determine objectively whether a product design is suitable for the RMS software product line.

Program Performance: Program performance will improve. Many current programs issue hundreds or thousands of unnecessary characters for each user keystroke. This document provides concrete guidelines that will assist you in solving this and similar performance problems.

New Product Integration: Software standards will also result in a more graceful expansion of the software product line. All new products will conform to a uniform set of interactive standards. It will be easier to integrate them into the product line.

VISTA-VIEW™ Compatibility: Standardized software behavior paves the way for important new products, most notably the VISTA-VIEW context management system. Since the programs VISTA-VIEW manages will behave similarly, integration will be simplified.

International Transportability: In combination with the new keyboard, the interactive model will enable DATAPOINT to ensure international transportability of our software. This will make us more competitive in the international marketplace. A standardized software environment will also make us more competitive domestically.

Documentation and Training: Finally, we will find it less time-consuming to document new products and train customers in their operation.

Software Design Objectives

DATAPOINT's interactive software model is based on a coherent design strategy. It has been developed in consultation with software, hardware, marketing, customer support, and documentation personnel.

It is beyond the scope of this document to convey an entire philosophy of software design. However, six specific guidelines evolved while the interactive model was being designed, and understanding these guidelines is instrumental to understanding the interactive model itself. This section explains these guidelines; you may want to read it before you read the specific standards contained in the rest of this document.

Support Real-Time Interaction

Interactive software gives the user the impression that he is *directly* manipulating the objects and data with which the program is concerned.

To solve a problem with a batch or batch-like facility, you generally must take at least two separate steps. First, you must configure, or set up, the problem in such a way that the batch facility can understand it. Then you run the batch facility to solve the problem as configured. Normally, you use such a facility to derive a static result (like "657") or to produce some particular effect (like printing a 300-page report).

Interactive facilities, on the other hand, let you solve problems dynamically. Rather than configuring a problem, you can explore and modify an information environment in real time.

Comparing the RMS CONFIG and STARTUP utilities is instructive. CONFIG is a batch facility. When you invoke it, you must answer a series of questions about the cofiguration file you want to create. Then, when you think you have properly constructed the configuration file, you copy it to the RMSAUTOSTART catalog on your booted resource and reboot your node. If you make an error in constructing the configuration file, you may not realize this until after you reboot, and you must then go through the entire process again. You cannot, in other words, reconfigure a node dynamically in real time.

The STARTUP utility is far more interactive. Its purpose is quite simple. Once you have dynamically established the environments you require (using the ENV or SCOUT utility), you run STARTUP to record what you have done. What makes STARTUP an interactive utility is that you ''configure'' your startup file by building it dynamically rather than by answering questions in a static environment, as with CONFIG.

Control the User's Point of View

Interactive software always deals with the user's point of view.

Programs in a ready state always display a single blinking system cursor. The cursor always appears at the position which currently requires the user's attention. Programs are free to change the user's focal point but should do so only by changing the position of the cursor. For example, as you enter text in IEOS™ the cursor moves along with the text as you enter it. When you press the Command Key to enter a command, the cursor moves to the command line.

In screen dialogue, programs should deal with directional input and output from the user's perspective. For example, when a user presses the Right Arrow Key, he indicates his desire to move his perspective to the right. This is sometimes accomplished by moving the cursor to the right, as in EASL, OOZL, and WP; at other times, the cursor remains in place while the text moves to the left beneath it, as in VISTA-VIEW. Screen dialogue must also refer to directions from the user's perspective. For example, a help message should use the term ''scroll up'' to indicate moving the user's perspective up — and thus moving the text down.

Avoid the Teletype® I/O Model

Interactive software prefers full-screen, runtime-adaptable screen interactions and avoids the teletype model.

An interactive facility can use the workstation screen in a variety of ways. Traditionally, DATAPOINT software has used the teletype model for screen interactions. The teletype model is fine for programs whose interaction is entirely defined by a single command line. However, the constraints it imposes on software designers render it useless in designing interactive software.

For example, the teletype model works well for the ENV utility when you invoke it to obtain a listing of your current environments. Inserting environments, however, is not well suited to the teletype model; SCOUT does a far better job of supporting dynamically interactive environment management.

To replace the teletype model, DATAPOINT has developed a general screen layout called the *RMS Dashboard*. The Dashboard provides an ideal backdrop for implementing the four techniques that

Teletype is a Trademark of Teletype Corp., Skokie, Ill.

comprise the interactive model. It is discussed in detail in Chapter 2, ''General Standards.''

Create a Single Interactive Context

Interactive software incorporates its entire set of capabilities into a single interactive context.

This guideline has two significant implications:

- A product that provides layered capabilities (to serve different user groups, for example) should do so within a single interactive context.
- A product that requires users to configure some aspect of its operation should integrate the configuration and operation modes in such a way that they are procedurally indistinguishable.

Layered Capabilities: A single software product often must serve several different groups of users. These groups frequently have different backgrounds, different levels of experience, and different needs.

In designing the VISTA-GUIDE™ component of the RMS 2 command interpreter, for example, it was necessary to serve both the naive end user, whose computer experience is minimal and likely to stay that way, and the RMS network administrator, whose experience is substantial and likely to increase. DATAPOINT determined it would answer the needs of both groups in a single interactive context. VISTA-GUIDE allows a network administrator to provide VISTA-GUIDE users with different capabilities by assigning security attributes to the files that support a user's VISTA-GUIDE context. However, these capabilities are implemented within a single interactive framework.

Distinctions between different classes of users do not justify developing separate interactive environments.

The filing clerk, the technical writer, and the editor of the corporate newsletter must be equally at home using DATAPOINT's word processing software. The facilities that it provides for high-end users cannot negate the simplicity and ease of use for low-end users. The best solution is to design the product to support several layers of capabilities.

Integrating Configuration and Operation: Certain desirable software features cannot be implemented without requiring users to configure some aspect of their operation. For example, a product cannot support user-programmable function keys without requiring the user to program them. We cannot eliminate configuration altogether, but interactive programs should allow users to configure an activity by actually performing it.

Comparing the IEOS function keys with the EASL macro facility is instructive. Before using a function key in IEOS, you must enter a syntactically correct command to configure the definition you wish to assign to it. If you have used the IEOS function key facility, you may have noticed that it can take several minutes of experimentation to get the configuration correct. The source of the difficulty lies in the fact that the configuration process requires you to think in stereo — tracking the keystrokes as you enter them *and* the effect they would have if you were actually entering them as instructions to IEOS rather than as the configuration of a function key.

The EASL macro facility, on the other hand, allows you to record a macro by actually performing the activity you want the macro to perform. You tell EASL that you want to record a macro, and EASL begins to record your keystrokes. You then do the activity you want to store as a macro and tell EASL when you have finished. Although you have entered a configuration, you have done so in exactly the same way that you would have performed the activity manually. Configuring a macro and performing the task manually are procedurally indistinguishable.

Avoid Hard Mode Changes

Interactive software avoids hard mode changes within an interactive context.

Simply defined, a *mode* is the set of capabilities a program makes available to its users at any given time. From time to time, programs may need to change modes. For example, when you press the Command Key in IEOS, you enter command mode and cannot perform editing functions until you return to editing mode by pressing the Return Key. In IEOS pressing the Command Key places you in a *hard mode*.

Programs that frequently change a user's set of capabilities in non-obvious ways tend to be confusing and disorienting. Programs should make any hard mode change visible on the screen so that a user knows his capabilities have changed. In IEOS, for example, you know you are in command mode because the cursor moves to the command line, and you know you are in insert mode because the word INSERT appears in inverse video on the information line.

There are several standard hard modes. Ring menus, for example, are a hard mode and are the standard method for command and option specification. The nature of command/option specification differs intrinsically from normal program operations, and ring menus enable command specification to function the same way in all interactive products.

Generally, however, interactive software should avoid hard mode changes within a single interactive context. One way to accomplish this is to support *key chords*. A key chord is simply a combination of keys which, when depressed at the same time, issues a single instruction to a program.

Key chords are already quite common. For example, to type a capital letter, you normally chord the Shift Key with the letter you want. To associate a command with an item in the VISTA-GUIDE hierarchy, you chord the Insert and Command Keys; to insert a help message, you chord the Insert and Help Keys. Because key chords do not create a program state that persists when you remove your hands from the keyboard, they are considered *soft mode* changes.

Because of DATAPOINT's new expanded function keyboard, you can use key chords to accomplish the same objectives you might previously have implemented through hard mode changes. The new keyboard will support key chords with any of the keys. This will allow WP to implement express cursor movement through chording the Window Key with cursor movement keys, thus avoiding the undesirable hard mode change.

Optimize Program Performance

Interactive software is optimized to enhance program performance to the maximum extent possible within current hardware limitations.

No one argues against the importance of program performance, but it is frequently underestimated as a factor in product acceptance. Simply stated, unacceptable performance can ruin a product's chance for success in the marketplace.

Program performance is increasingly important at DATAPOINT because of architectural and software changes that will impose significantly greater overhead on user programs. VISTA-VIEW, CPF, and ATTACH slow response time user programs. Programs also perform less efficiently through a local or dial-up serial port. And finally, this extra overhead often slows the performance of other workstations when one abuses the processing power of the computer. There are a number of techniques you can use to optimize your programs to run well under such circumstances. These techniques are discussed in Chapter 7, ‘‘Technical Considerations and Standards.’’

NOTES

CHAPTER 2.
GENERAL STANDARDS

Contents

The RMS Dashboard

The dashboard in your car has a number of meters, lights, and gauges that provide you with information you need in order to drive efficiently and safely. The information always appears in the same place. Imagine how confusing and annoying it would be if the speedometer was in a different place each time you got into your car.

The four techniques that comprise the interactive model work together within a general screen layout called the *RMS Dashboard.* The RMS Dashboard provides the information you need in order to ''drive'' your workstation. Its design ensures that the same information always appears in the same place, no matter what application you are using. This consistency permits users to concentrate fully on the work they are doing; they are not visually distracted as they migrate from one program environment to another.

The VISTA-GUIDE™ and VISTA-SCRIPT™ components of the RMS 3 command interpreter exemplify the RMS Dashboard design. VISTA-GUIDE and VISTA-SCRIPT will provide the only framework through which a significant number of our RMS users interact with the operating system. It is therefore sensible to develop an RMS 2 software environment based on the RMS Dashboard.

All future RMS software will work within the RMS Dashboard (Figure 2-1). It consists of four screen elements:

- the Banner Line,
- the Frame,
- the Option/Prompt Line, and
- the Help/Response Window.

Programs that are primarily concerned with freeform text editing often have certain special requirements, including a larger displayable area than the Dashboard allows. There is an alternate Dashboard format that allows for the special requirements of such programs. This alternate format is described at the end of this chapter.

The Banner Line

The Banner Line occupies the first screen line and displays a program identification message. The message is left-justified at column 1 and appears in normal video.

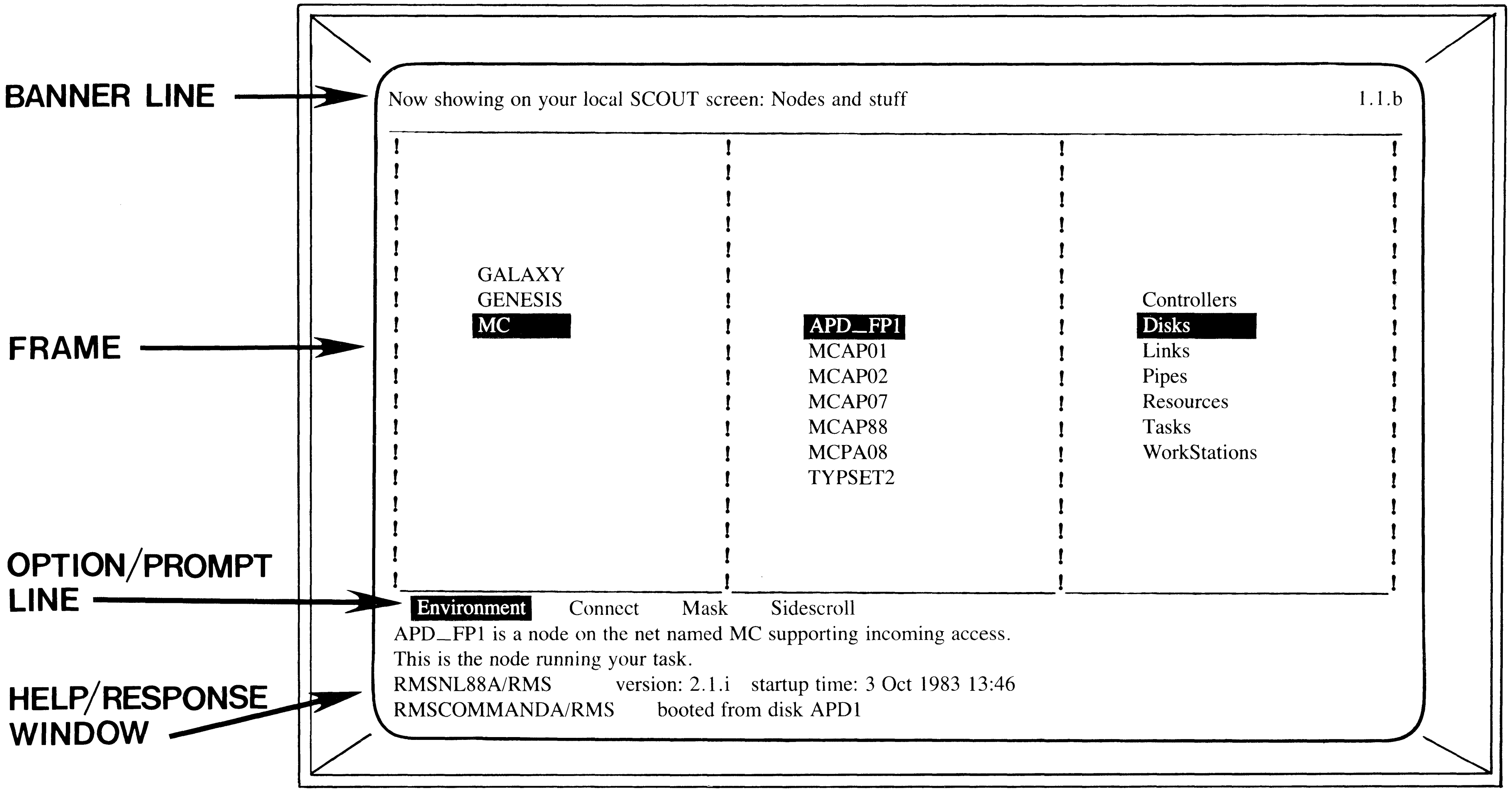
Most applications are free to display a message from positions 1 through 79; however, Miller column applications that provide editing services must reserve positions 27 through 80 for screen animation relating to a recallable LIFO stack. Detailed information about this screen animation is contained in Chapter 3, ‘‘The Miller Column Technique.’’

The Frame

The Frame is a bordered box that occupies screen lines 2 through 19. Most routine program activities occur within the Dashboard Frame.

The Frame is left-justified at column 1. The box top appears on screen line 2; the box bottom appears on screen line 19. The sides of the box are drawn in columns 1 and 79. Column 80 is not used. The Frame contains a displayable area of 16 lines by 77 columns.

Programs that utilize the teletype model for screen interactions should use the Dashboard Frame in the same manner they have traditionally used the full workstation screen; that is, teletype interactions, such as those in the INFO and SECURE utilities, should take place entirely within the Frame. Available subwindowing facilities enable you to use the full



```
Now showing on your local SCOUT screen: Nodes and stuff                    1.1.b

!                      !                      !                      !
!                      !                      !                      !
!                      !                      !                      !
!                      !                      !                      !
!                      !                      !                      !
!     GALAXY           !                      !                      !
!     GENESIS          !                      !    Controllers       !
!     MC               !      APD_FP1         !    Disks             !
!                      !      MCAP01          !    Links             !
!                      !      MCAP02          !    Pipes             !
!                      !      MCAP07          !    Resources         !
!                      !      MCAP88          !    Tasks             !
!                      !      MCPA08          !    WorkStations      !
!                      !      TYPSET2         !                      !
!                      !                      !                      !
!                      !                      !                      !
!                      !                      !                      !
!                      !                      !                      !
  Environment   Connect   Mask   Sidescroll
APD_FP1 is a node on the net named MC supporting incoming access.
This is the node running your task.
RMSNL88A/RMS        version: 2.1.i   startup time: 3 Oct 1983 13:46
RMSCOMMANDA/RMS       booted from disk APD1
```

Figure 2-1. The RMS Dashboard

capabilities of the workstation screen within the Dashboard Frame.

The Option/Prompt Line

The Option/Prompt Line occupies columns 1 through 79 on screen line 20. It is reserved for ring menus and associated prompts.

The Help/Response Window

The Help/Response Window occupies screen lines 21 through 24 from column 1 through 79. Programs may use it to display context-sensitive help messages and to collect user input relating to program operations or ring menu interactions.

The Continuation Bar

Many users will only see RMS through VISTA-GUIDE, which, like all Dashboard programs, uses the full workstation screen. Each time a user wants to run a program, he will locate the appropriate activity in the VISTA-GUIDE hierarchy and then invoke the program by pressing the Return Key. When the program terminates, it will return control to VISTA-GUIDE so that the user can select the next activity he wants to run.

With certain types of programs, users need the ability to pause before returning to VISTA-GUIDE. For example, after selecting the ''List file names'' activity in VISTA-GUIDE, the following series of events might occur:

- control passes from the command interpreter to the CAT utility;
- the utility reads the selected catalog and displays file names on the screen in accord with any parameters the user specified;

- control returns to the command interpreter, which erases the file names from the screen and repaints the VISTA-GUIDE hierarchy.

The problem here is that the user never got the chance to read the file names. The RMS Dashboard supports a *continuation bar* to resolve this problem.

Any program that displays data for its users but does not provide them with runtime, interactive control should display a continuation bar upon normal program termination for any user running under VISTA-GUIDE. Actually, the command interpreter displays the continuation bar, depending on the contents of the $PCRCMDF flag (explained further below).

The continuation bar is an inverse video strip occupying columns 1 through 79 on screen line 24. The following message appears centered within the continuation bar:

■ Press any key to continue

The cursor appears one position to the left of the capital *P*.

Some programs support interactive environments of their own. These programs normally support the Quit Key to allow users to return voluntarily to the command interpreter and therefore should not display a continuation bar. For example, MAIL, SCOUT, IEOS, and Multiplan™ should not display continuation bars, whereas ENV, CAT, and INFO should.

NOTE: No program should ever display a continuation bar when control passes to the command line interface of the command interpreter. Continuation bars only appear for users running under VISTA-GUIDE. To suppress the continuation

Multiplan is a Trademark of Microsoft, Inc.

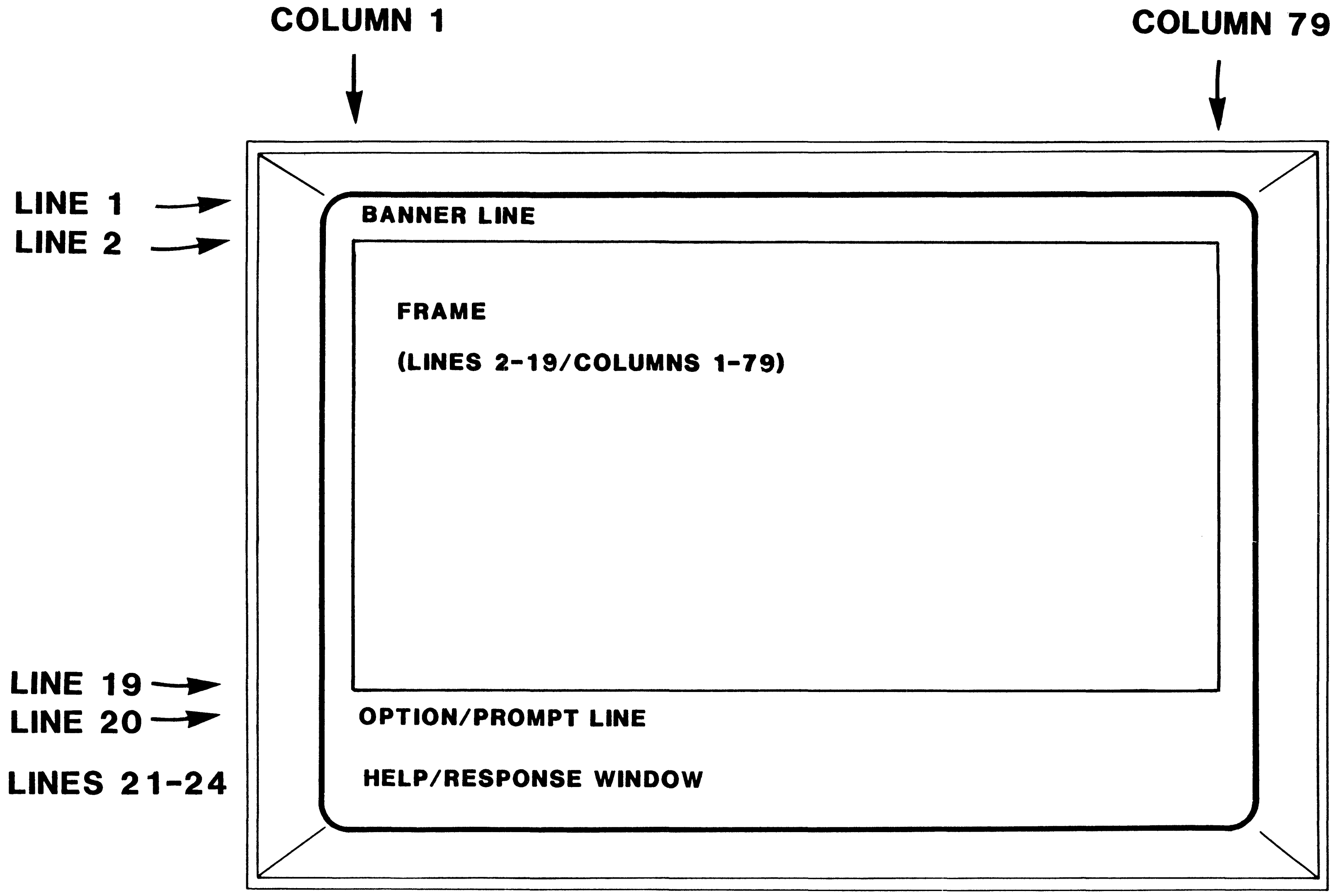


Figure 2-2. A More Detailed Look at the Dashboard

bar, set on the $PCRDFNS bit in the $PCRCMDF flag.

Transitions Between Dashboard Programs

Whenever control passes from one program to another, the newly invoked program is responsible for ensuring a visually smooth transition. The new program's first actions depend upon whether the program from which it assumes control conforms to the Dashboard design described above.

Upon being invoked, a program should read the $PCRCMDF flag in the PCR to determine the type of program from which it is assuming control. The visual nature of the transition depends upon the contents of the two bit flags:

- **$PCRDFNS:** If this bit is on, the previous program was a Dashboard program, and it is leaving data on the bottom 4 screen lines which the newly invoked program should not erase.
- **$PCRCMDF:** If this bit is on, the previous program was a Dashboard program, and it is surrendering the entire screen.

If neither of these bit flags is set on, you may assume that the previous program was not Dashboard-based.

Assuming Control from a Dashboard Program

If either the $PCRDFNS or $PCRDFNC bits in the $PCRCMDF flag are set on, the new program is assuming control from a Dashboard program. The new program should assure a smooth transition by supporting the following animation (See Fig. 2-2):

1. If the $PCRDFNC bit is set on, erase everything on the screen except for the box-drawing characters. If the $PCRDFNS bit is set on, erase everything on the screen except for

the box-drawing characters and the bottom 4 lines of the screen.

2. If the new program is a Miller column application, divide the Frame into three equal columns. To do so, first rewrite the characters in columns 27 and 53 on screen line 2, then paint vertical lines in those columns through screen line 18, and finally rewrite the Frame characters in columns 27 and 53 on screen line 19.

3. Write the contents of the Banner Line.

4. Write the contents of the Frame. For a Miller column application, write the contents of the three columns — first the left column, then the center column, then the right column.

5. Write any information that appears in the Option/Prompt Line and Help/Response Window.

Assuming control from a Non-Dashboard Program

If both the $PCRDFNS and $PCRDFNC bits in the $PCRCMDF flag are set off, the new program is assuming control from a program that does not conform to the standard RMS Dashboard model. In this situation, it is not possible to ensure a visually smooth transition, but the new program should support the following animation:

1. Erase everything on the screen.

2. Write the Banner Line.

3. Write the Frame and its contents line by line. (Write screen line 2, then screen line 3, then screen line 4, etc.)

4. Write any information that appears in the Option/Prompt Line and Help/Response Window.

The Alternate RMS Dashboard

Programs that are primarily concerned with freeform text editing, including word processing programs, often have as a major activity displaying as much data at once as possible, for example displaying as much of a document as possible. The standard RMS Dashboard format is not ideal for such programs. You may use the alternate format described below for programs whose primary concern is displaying freeform data.

The alternate format entails a simple rearrangement of the Dashboard elements, as follows:

- **Frame:** The Dashboard frame occupies screen lines 1 through 19 from column 1 through column 80. There are no box-drawing characters, and the entire area is available to the application program.
- **Banner Line:** Screen line 20 appears in inverse video from column 1 through column 80 and contains the program identification message that would appear in the Banner Line in the standard RMS Dashboard.
- **Option/Prompt Line:** The Option/Prompt Line appears on screen line 21 and functions as described earlier.
- **Help/Response Window:** The Help/Response Window appears on screen lines 22 through 24 and functions as described earlier.

The Cursor

Ready-state programs always display a single blinking system cursor, which serves three purposes.

Indicates Current Program State

The cursor indicates whether a program is currently prepared to accept user input. Whenever a program is listening to the keyboard, the cursor appears on the screen; whenever it is not, it turns the cursor off.

Indicates User's Focal Point

DATAPOINT's interactive software must always deal with the user's point of view. Therefore, all programs must display the cursor in the position that is the user's current focal point. (In field keyin applications, the cursor must always be positioned in the field and at the precise location into which the user is entering data.) This is clearly a requirement when subwindows are used in VISTA-VIEW, as explained below.

Indicates VISTA-VIEW Focal Point

No program may ever display more than one cursor at a time. VISTA-VIEW uses the cursor position to determine the part of a full-screen display visible.

For example, if the cursor appears in the Frame while the program is waiting for the user to select a ring menu option, the user may not even be able to see the ring menu through the VISTA-VIEW window he has created. However, if the cursor moves from the Frame to the ring menu when the user invokes the ring menu, VISTA-VIEW will reposition the display so that the ring menu is visible through the VISTA-VIEW subwindow.

The RMS 2 Keyboard

DATAPOINT has created a new keyboard, called the RMS 2 keyboard, that supports the interactive software model more efficiently than the current general purpose keyboard. A number of new features have been designed into the RMS 2 keyboard, including:

- dedicated editing keys,
- dedicated context switching keys,
- LCD status indicators,
- a new software interface,
- eight program-definable function keys,
- separate cursor movement and numeric entry keypads,
- alternate shift keys, and
- several new program control keys.

Because of certain software and hardware compatibility concerns that accompany the introduction of a new keyboard, the RMS 2 keyboard operates in two different modes.

Mode One: In mode one, the RMS 2 keyboard simulates the current general purpose keyboard (See Figure 2-3). It generates only those codes that can be generated by the general purpose keyboard.

Mode Two: In mode two (See Figure 2-4), each keyswitch generates one unique code when pressed and another unique code when released. A layer of the RMS nucleus known as OPTIK serves as the interface between the keyboard hardware and user-level software.

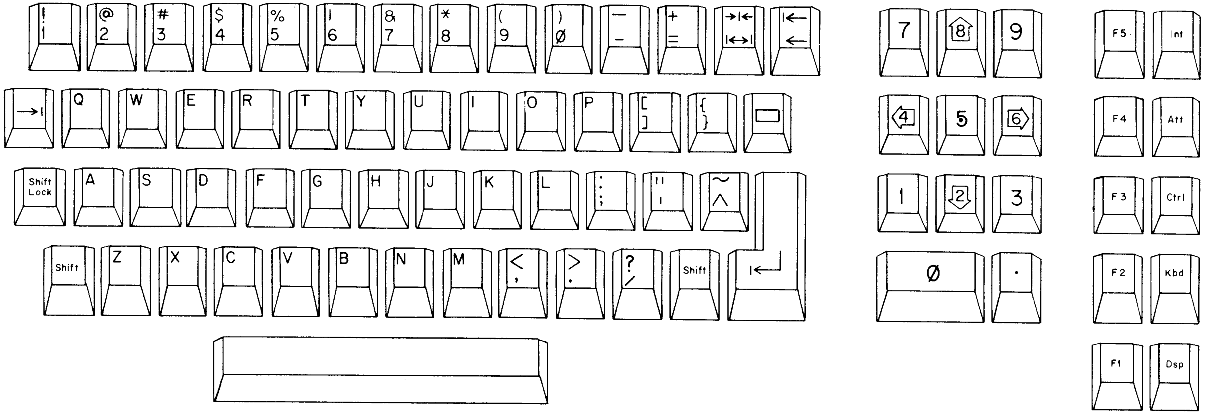


Figure 2-3. The Old General Purpose Keyboard

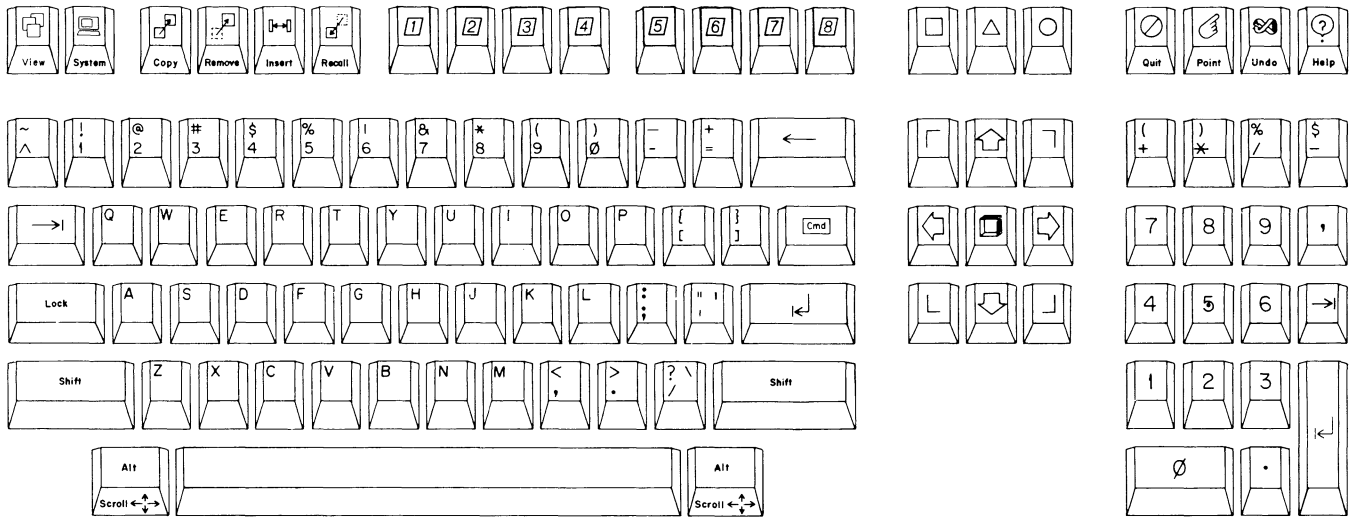


Figure 2-4. The RMS 2 Keyboard

Keys on the Keyboard

Certain standards apply to the manner in which interactive software uses the keys on the RMS 2 keyboard. These standards are described below.

Alphanumeric Keys

These keys generate printable characters in the range of 02-176 (octal). Programs may not implement program control sequences that use printable characters, except for such standard program mechanisms as ring menus and command lines which are message member translatable.

Alternate Shift Keys

The Alternate Shift Keys (marked Alt) provide a level of differentiation beyond the Shift Keys. There is a lowercase A, an uppercase A (ShiftA), and an alternate A (AltA). The same holds true for each key on the keyboard.

Currently, the only USA functions of the Alt Keys are: scrolling, when combined with certain cursor movement keys; and, deletion of imbedded characters, much as the shift/delete works on the general purpose keyboard. Chording either Alt Key with any arrow key causes scrolling to occur in the indicted direction. For example, AltLeft moves the user's perspective left (and the text right); AltUp moves the user's perspective up (and the text down). There is no definition for the chords formed by the Alt Key and the corner keys or the Window Key, and until they are formally defined by DATAPOINT, user programs should avoid recognizing these chords.

Backspace Key

The Backspace Key causes the cursor to move backward, deleting each character it moves over. The

Backspace Key does not allow users to move from one field to another in field keyin applications; cursor movement keys support this function. The Alt/Backspace Key has the same effect as the shift/delete key on the general purpose keyboard.

Command Key

The Command Key toggles a user in and out of *command mode*, in which he uses a ring menu to select or specify options relating to the activity in which he is currently involved.

Function Keys

The RMS 2 keyboard has eight program-definable function keys. Applications should avoid using these keys to provide command/option specification; this is correctly done through the ring menu technique. However, application developers may incorporate function keys into a new product so long as the implementation does not violate any of the guidelines or standards presented in this document.

Geometric Keys

These three keys, marked by a circle, a triangle, and a square, will be used to integrate certain context management mechanisms that are still under design. Application programs should not use these keys or the corresponding LCD indicators on the keyboard.

Help Key

The Help Key provides the user with context-sensitive, clearly written help messages, which normally appear in the Help/Response Window.

Numeric Keypad

On the numeric keypad, the number keys and the comma generate the printable characters and functions that appear on the key caps.

In mode one, the Tab and Return Keys, along with the four keys on the top row, generate the printable characters that appear on the key caps. In mode two, however, these six keys generate unique codes. Applications may elect to treat these keys as identical to those on the alphanumeric keypad, or they may elect to alter processing depending on which key is pressed.

To ensure compatibility between the current general purpose keyboard and the RMS 2 keyboard, the zero and decimal point keys on the numeric keypad of the current general purpose keyboard must be recognized as an ASCII zero and period. This requirement applies to all RMS and DOS software under all circumstances.

Quit Key

The Quit Key is the standard method for allowing a user to leave a context. This context may be a program (such as SCOUT) or a context within a program (such as a help message).

Programs that do not involve writing data should return control to the next program to be run (usually the command interpreter) whenever a user presses the Quit Key. On the other hand, if a user might lose data by exiting accidentally, a program should respond to the Quit Key by displaying a ring menu that includes two options: ''Save'' and ''Discard.'' The user can then indicate whether or not he wishes to retain the data he has entered.

The Quit Key is also the standard mechanism for interrupting a process that is not currently listening to the keyboard. For example, the Quit Key should terminate a search and replace operation, a Place command in IEOS, or the recalculation of a Multiplan spreadsheet. The KBD Key is used to serve this purpose; however, the RMS 2 Keyboard does not have a KBD Key, and the KBD Key on the current keyboard is now used for scrolling (and is often called the Upscroll Key).

Shift/Shift Lock Keys

The current general purpose keyboard supports shift lock. In mode one, the RMS 2 keyboard also supports shift lock. However, in mode two, users who press the Lock Key will be placed in caps lock mode.

Program controls should always operate without respect to case. For example, IEOS does not require users to enter commands in upper case; they can use upper case, lower case, or any combination of the two. All programs should support this type of case insensitivity.

System Key

User programs should not use the System Key for any purpose. It is the basis of a number of key chords that send instructions directly to the RMS nucleus. Any time the System Key is pressed, all subsequent keystrokes are processed by the nucleus until the System Key is released. Consequently it is not available to user programs.

Tab Key

The Tab Key should only be used for tabbing in freeform text editing applications. It does not allow users to advance from one field to the next in field keyin applications; cursor movement keys support this function.

Undo Key

The Undo Key acts as a toggle switch, reversing the effect of an action the user has taken when such an action is not reversible through any other obvious mechanism. In IEOS, for example, if you press the word forward key, you can easily reverse the action by pressing the word backward key; since the word forward action is easily reversible, IEOS need not support it through the Undo Key. In VISTA-SCRIPT, on the other hand, accidentally typing a single letter can change the entire composition of an activity script; there is no obvious way to reverse the action, so pressing the Undo Key returns the script to its former state.

Specific functions of the Undo Key have been defined for field keyin, but not for the other techniques. Application developers may use the Undo Key in application-specific ways that are consistent with the general guideline presented above.

View Key

Only programs that perform global context management should use the View Key. Currently, the only such program is VISTA-VIEW. The View Key is the basis of a number of key chords that send instructions directly to the VISTA-VIEW program. Any time the View Key is pressed, all subsequent keystrokes are processed by VISTA-VIEW until the View Key is released. Consequently this key is not available to any program that might operate under VISTA-VIEW; currently, this includes all programs.

Keystroke Recording

Ensuring the recoverability of terminal sessions is a critical aspect of developing reliable, quality software. RMS programs that let users write data must support work session recoverability. As a minimum, some programs may implement keystroke recording as a means to this end.

In evaluating the suitability of keystroke recording, however, you must consider the possibility that multiple users will access the same information simultaneously, as with a shared database. In such cases, recovery may well not be possible with a mechanism as simple as keystroke recording.

However, keystroke recording is adequate in many situations. For example, a word processing package, an electronic spreadsheet, and a text editor all may support work session recoverability through keystroke recording.

Keystroke recording is a simple process. When a program is invoked, it creates a special log file and begins to write each user keystroke into the log file. If the program terminates cleanly, it deletes the log file. If it terminates abnormally, however, a complete reconstruction of the work session is possible.

Synchronous Recording

Certain types of program actions are not obviously reconstructable — text scrolling, for example. It is not sufficient to know that the user pressed the Downscroll Key for six seconds. A different level of system overhead during recovery will result in a different amount of scrolling within the same time period.

Each program must assume responsibility for ensuring that the events recorded in its log file are deterministically repeatable. This means, for example, that the log file needs to know exactly how many lines of text the user scrolled through.

Log File Housekeeping

Some programs must erase their log files periodically to ensure that completed tasks are not re-executed when the user attempts to recover a work session. Electronic mail provides a useful example.

In a typical RMS Mail work session, you might create and send a memo, then start to create a second memo. At this point, the system may crash. While it is desirable to be able to recover the work in progress on the second memo, you do not want the recovery function to regenerate and resend the first memo; that task was already successfully completed. In applications like RMS Mail, the keystroke recording mechanism must make intelligent decisions about when to clear its log file.

CHAPTER 3.
THE MILLER COLUMN TECHNIQUE

Contents

General Principles

The Miller column technique allows programs to represent hierarchically structured information in a graphically interactive context. The technique consists of

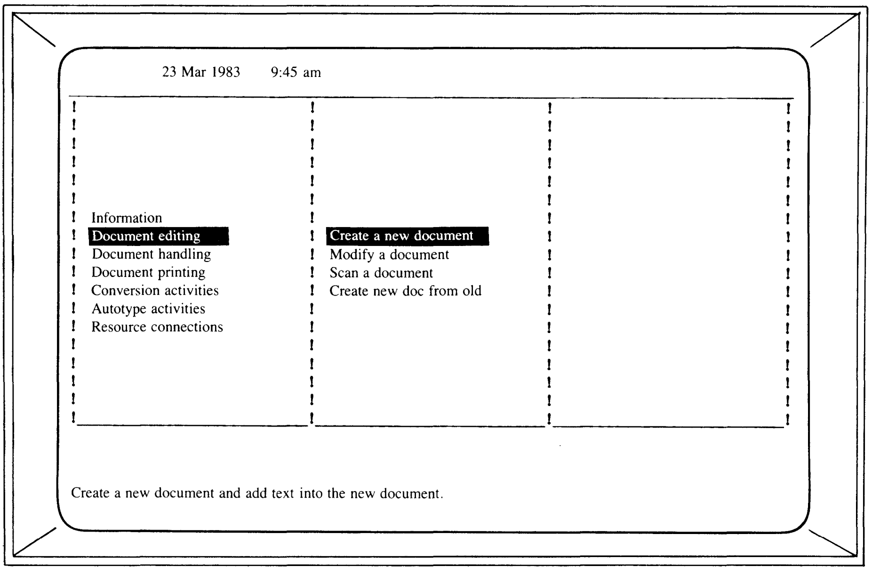
- a screen layout,
- definitions for eight scrolling keys, and
- definitions for nine editing operations.

The VISTA-GUIDE component of the RMS 2 command interpreter lets you roam around a Miller column presentation of activities you might want to perform with your RMS system. SCOUT uses the Miller column technique to represent the files and resources available through an RMS network.

The Miller column technique divides the Dashboard Frame into three columns of equal width. Each column may contain a series of 23-character, alphanumeric *items* which represent in a hierarchical pattern the objects or data with which the program is concerned.

An example of the VISTA-GUIDE hierarchy appears below. Notice that the three columns are linked hierarchically. Two items always appear in inverse video — one in the left column and one in the center column. The highlighted item in the left column is the hierarchical ancestor of the highlighted item in the center column. The items in the right column are the hierarchical descendants of the highlighted item in the center column.

In the VISTA-GUIDE hierarchy above, ''Create a new document,'' ''Modify a document,'' ''Scan a document,'' and ''Create new doc from old'' are the hierarchical descendants of ''Document editing'' (Figure 3-1).



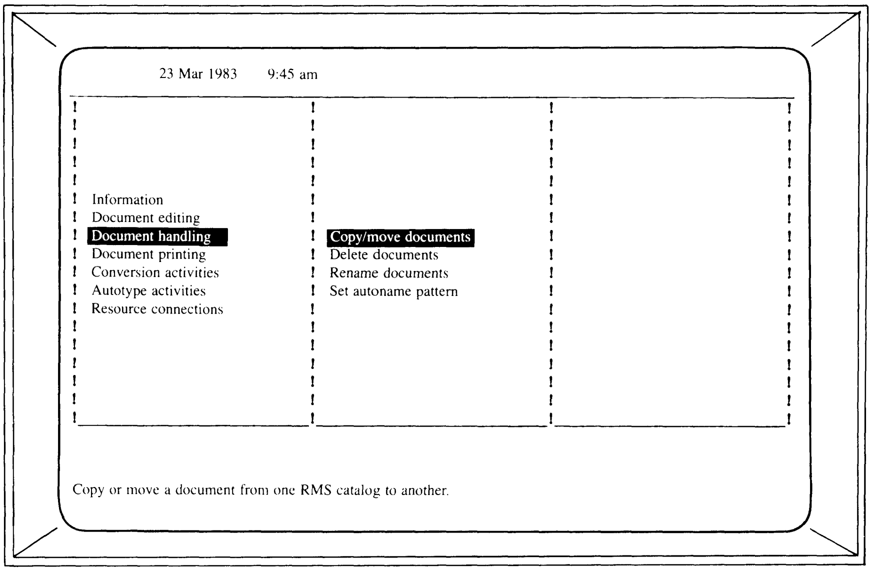
```
23 Mar 1983      9:45 am

Information
Document editing               Create a new document
Document handling              Modify a document
Document printing              Scan a document
Conversion activities          Create new doc from old
Autotype activities
Resource connections

Create a new document and add text into the new document.
```

Figure 3-1.   The RMS Dashboard

The Miller column technique supports eight scrolling keys, which you can use to move around in the hierarchy. If you use one of these keys to highlight a different item in the center column, new items may appear in the right column. Similarly, if you highlight a different item in the left column, new items may appear in both the center and right columns. Look what happens when you change the current item in the sample VISTA-GUIDE screen (Figure 3-2):

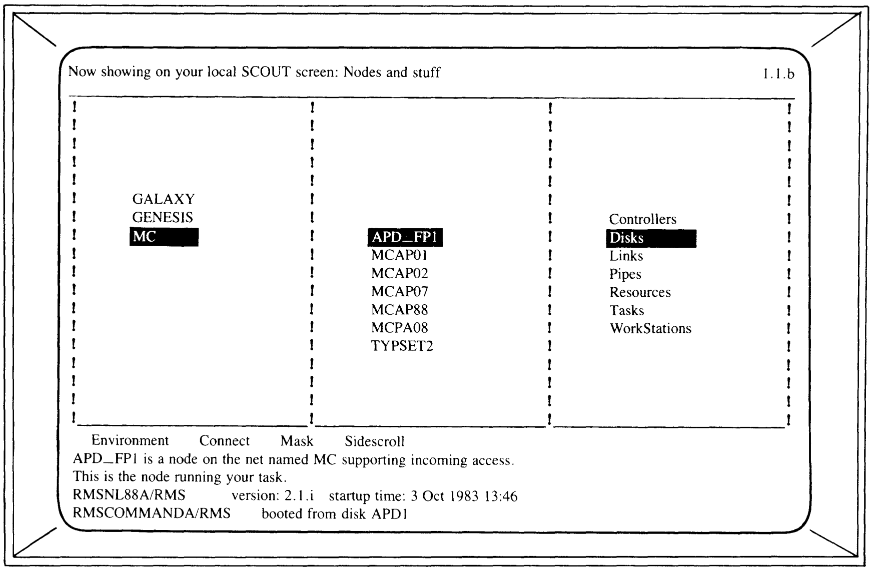

23 Mar 1983 9:45 am

Information		
Document editing		
Document handling	Copy/move documents	
Document printing	Delete documents	
Conversion activities	Rename documents	
Autotype activities	Set autoname pattern	
Resource connections		

Copy or move a document from one RMS catalog to another.

Figure 3-2. The RMS Dashboard

Item is not a very descriptive word, but this is deliberate. The precise nature of the items that appear in a Miller column display depends upon the objectives and requirements of each application. For example, SCOUT, a network management aid, uses items that represent the hierarchical elements in an RMS network (Figure 3-3):



```
Now showing on your local SCOUT screen: Nodes and stuff                          I.I.b

!                  !                    !                     !
!   GALAXY         !                    !                     !
!   GENESIS        !                    !   Controllers       !
!   MC             !   APD_FP1          !   Disks             !
!                  !   MCAP01           !   Links             !
!                  !   MCAP02           !   Pipes             !
!                  !   MCAP07           !   Resources         !
!                  !   MCAP88           !   Tasks             !
!                  !   MCPA08           !   WorkStations      !
!                  !   TYPSET2          !                     !
!__________________!____________________!_____________________!
   Environment   Connect   Mask   Sidescroll
APD_FP1 is a node on the net named MC supporting incoming access.
This is the node running your task.
RMSNL88A/RMS       version: 2.1.i   startup time: 3 Oct 1983 13:46
RMSCOMMANDA/RMS      booted from disk APD1
```

Figure 3-3. The RMS Dashboard

Any application that needs to represent a hierarchical information structure is probably well-suited to the Miller column technique.

Benefits of the Miller Column Technique

The ability to explore freely helps the user to make sense out of what may in fact be a complicated information hierarchy. The ability to edit the hierarchy allows users to manipulate the objects represented in the hierarchy and to tailor application software to their own requirements.

The Miller column technique constantly provides a visual indication of your position within a hierarchy and thus within the task you are performing. In VISTA-GUIDE, for example, you can see your current set of alternatives in the center column, with the preceding level and the next generation to either side. This prevents you from becoming disoriented in a complex information environment.

Finally, since even the best application developer cannot anticipate the needs of every user, the editing model for Miller columns lets users tailor hierarchies to suit their own particular needs.

What Miller Columns Look Like

The screen layout for Miller column applications has been standardized and is presented here to ensure cross-application uniformity.

Miller Columns in the RMS Dashboard

The Miller column technique has been designed to integrate into the RMS Dashboard described in Chapter 2, ''General Standards.'' It makes the following use of its four elements:

Banner Line: The Banner Line holds a standard program identification message. However, if a Miller column application supports editing capabilities, columns 27 through 80 of the Banner Line are reserved for the animation of activities relating to the LIFO stack. This animation is discussed later in this chapter.

Frame: In Miller column applications, the Frame serves as a border drawn with box-drawing characters, as discussed in Chapter 3.

Option/Prompt Line: The Option/Prompt Line is reserved expressly for ring menus. Many of the standard Miller column editing techniques involve the use of ring menus. Applications which support these techniques must do so through standard ring menus. Applications which do not support these techniques should not use the Option/Prompt Line.

Help/Response Window: Miller column applications should use the Help/Response Window to display context-sensitive help information or to collect user input relating to program operations or ring menu interactions.

Dimensions of the Miller Column Display

The Dashboard Frame is modified by the addition of vertical lines in columns 27 and 53. These lines divide the Frame into three Miller columns, each of which contains a 16 line by 25 character displayable area. The positions at which these vertical lines intersect with the Frame borders are occupied by T and inverted-T graphic characters.

Item Padding Spaces

Each item always appears with a blank space to its right and left. When an item appears in inverse video, so do these *padding spaces*.

Inverse Video

In the left and center columns, the items that appear on the tenth screen line always appear in inverse video (along with their padding spaces). Items in the right-hand column never appear in inverse video.

Cursor Behavior

The highlighted item in the center column is called the *current item*. It is the user's focal point. The system cursor always appears over the left-hand padding space of the current item, unless the user's focal point changes, for example to a ring menu on the Option/Prompt Line.

The Icon

Every Miller column application has associated with it an icon, which is an alphanumeric or graphic image suggestive of the application. For example, VISTA-GUIDE uses an icon suggestive of the DATAPOINT software world: the DATAPOINT "D".

The Miller column icon occupies the leftmost column in the hierarchy. Whenever the application is invoked, the Miller columns appear with the icon in the left column and the first data column in the center. An exception is Vista-Guide, which returns the user to the position the columns occupied when they were last displayed.

Moving Around in Miller Columns

One of the primary purposes of Miller columns is to allow users to roam around within an information hierarchy. There are standard definitions for eight scrolling keys that allow a user to change his position within a hierarchy.

These eight keys do not provide cursor movement. The cursor always appears over the current item unless the user is engaged in a ring menu or prompt on the Option/Prompt Line.

The following pages describe the screen animation associated with each of the eight scrolling keys. There are references throughout to columns changing position. Whenever the standards call for columns to move a column width to the left or right, what is called for is smooth scrolling, not a sudden repainting of the screen.

Left Arrow Key

The Left Arrow Key moves the Miller columns one column width to the right.

If the center column is already the topmost data column (i.e., the icon currently appears in the left column), the Left Arrow Key produces no effect. Otherwise, the Left Arrow Key changes the current item from inverse to normal video, then moves all three columns one column width to the right.

Right Arrow Key

The Right Arrow Key moves the Miller columns one column width to the left.

If the current item has no descendants, the Right Arrow Key produces no effect. Otherwise, the Right Arrow Key changes the item on the tenth screen line in the right column from normal to inverse video, then moves all three columns one column-width to the left.

Up Arrow Key

The Up Arrow Key moves the items in the center column down one line. If the current item is at the top of the column, the Up Arrow Key produces no effect.

Down Arrow Key

The Down Arrow Key moves the items in the center column up one line. If the current item is at the bottom of the column, the Down Arrow Key produces no effect.

Top Left Corner Key

The Top Left Corner (TLC) Key moves the items in the left column down one line. If no item appears above the tenth screen line in the left column, the TLC Key produces no effect.

If the user selects an item in the left column for which no descendants exist, thus leaving a ``black hole'' in the center column, all three columns move one column width to the right (as if the user had pressed the TLC Key followed by the Left Arrow Key).

Bottom Left Corner Key

The Bottom Left Corner (BLC) Key moves the items in the left column up one line. If no item appears below the tenth screen line in the left column, the BLC Key produces no effect.

If the user selects an item in the left column for which no descendants exist, all three columns move one column width to the right (as if the user had pressed the BLC Key followed by the Left Arrow Key).

Top Right Corner Key

The Top Right Corner (TRC) Key moves the items in the right column down one line. If no item appears above the tenth screen line in the right column, the TRC Key produces no effect.

Bottom Right Corner Key

The Bottom Right Corner (BRC) Key moves the items in the right column up one line. If no item appears below the tenth screen line in the right column, the BRC Key produces no effect.

Screen Update Timeout

When you change the highlighted option in the center or left column, other items in the hierarchy may change. Applications should not attempt to update the items in the hierarchy during the scrolling process; to do so would visually confuse users and impose unacceptable performance limitations. The standard timeout for Miller column updating is one-half second. This means that Miller column applications should update the items in the columns whenever a user ceases scrolling for one-half second.

Inverse Video During Scrolling

During vertical scrolling of the columns, the inverse video bars that mark the current item and its ancestor must remain solid. You can accomplish this by treating each column as a set of three separate subwindows:

- the region above the tenth screen line,
- the tenth screen line itself, and
- the region below the tenth screen line.

If your program first rewrites the tenth screen line, then scrolls the other two subwindows, the inverse video bars need never blink off.

Editing Operations

Some Miller column applications just allow the user to move around in, and thus investigate the structure and content of, an information hierarchy. However, most applications will want to provide some users the ability to edit the hierarchy — to add, delete, and change items and information that is associated with them. For example, one of VISTA-GUIDE's benefits is that it allows users to tailor the activity hierarchy to their own personal requirements.

The Miller column editing model allows for complex restructuring of the items in a hierarchy. It provides the user with facilities that enable him to enter new items, delete existing items, and move items from one position to another within the hierarchy. Central to the editing model is a last-in-first-out (LIFO) stack in which the user can temporarily store items that he is editing or repositioning within the hierarchy.

Miller column applications that support any of the editing capabilities discussed in this section must provide these capabilities as described here. For example, programs must use the Insert Key to support the Insert operation; this may not be implemented as a ring menu option. Application designers may of course develop application-specific program controls that are not discussed here.

Insert

The Insert operation allows the user to add an item to the hierarchy. The Insert operation is invoked by pressing and releasing the Insert Key — or, on the current general purpose keyboard, the F2 Key.

Screen Animation

When a user presses and releases the Insert Key, the following animation occurs:

1. The current item and the items above it in the center column move up one screen line, leaving a blank space in which the user may enter a new item from the keyboard.

2. The cursor appears in the first data column of the blank item, and the application waits for field keyin (which proceeds according to the standards presented in Chapter 5, ''The Field Keyin Technique'').

The inverse video bar remains constant throughout this operation. The reason for moving the current item up, rather than down, is to allow the user to enter a series of items in top-to-bottom order.

The Return Key: If the user presses the Return Key with the cursor over the first character position of the new item, or if he presses the Return Key when he has entered only blanks, no new item is inserted. Instead, the animation is reversed; that is, the items which originally moved up to make room for the new item move back down one screen line, and the current item is the same item that was current when the user invoked the Insert operation.

InsertLeft

The InsertLeft operation allows the user to create a new column between the left and center columns. The InsertLeft operation is invoked by chording the Insert Key with the Left Arrow Key.

Screen Animation

When a user presses and releases the InsertLeft key chord, the following animation occurs:

1. The current item changes from inverse to normal video.

2. While the left column remains stationary, the center and right columns move one column width to the right; the old center column becomes the new right column, and the old right column disappears off screen. This creates a blank center column into which the user can enter an item from the keyboard.

3. As soon as the new center column is three columns wide, an inverse video bar appears and stretches out to fill the new column.

4. The cursor appears in the first data column of the blank item, and the application waits for field keyin (which proceeds according to the standards outlined in Chapter 5, ''The Field Keyin Technique'').

The Return Key: If the user presses the Return Key with the cursor over the first character position of the new item, or if he presses the Return Key when he has entered only blanks, the new item and the new column must be discarded and the animation which created them reversed. In other words, with the left column stationary, the center and right columns move one column width to the left; the old right column becomes the new center column, and a new right column moves in from off-screen.

InsertRight

The InsertRight operation is the geometric opposite of InsertLeft. In other words, it allows the user to create a new column between the center and right columns. It is also the only way to extend the hierarchy to a new level. The InsertRight operation is invoked by chording the Insert Key with the Right Arrow Key.

Screen Animation

The animation that accompanies the InsertRight operation is similar to that for InsertLeft. While the right column remains stationary, the center and left columns move one column width to the left; the old center column becomes the new left column, and the old left column disappears off screen. This leaves a blank center column into which the user can enter an item from the keyboard.

The Return Key: If the user presses the Return Key with the cursor over the first character position of the new item, or if he presses the Return Key when he has entered only blanks, the new item and the new column must be discarded and the animation which created them reversed. In other words, with the right column stationary, the center and left columns move one column width to the right; the center column (which is vacant) disappears; the old left column becomes the center column again; and a new left column moves in from off screen.

Remove

The Remove operation allows the user to remove the current item from the hierarchy (along with any of its hierarchical descendants) and place it in a recallable LIFO stack. The Remove operation is invoked by pressing and releasing the Remove Key — or, on the current general purpose keyboard — the F3 Key.

Screen Animation

Generally, the following animation occurs when a user presses the Remove Key:

1. The current item disappears from its position, and the items above it in the center column move down one screen line.

2. The removed item reappears in inverse video aligned over the center column on the banner line. The program smoothly scrolls columns 28 through 80 of the banner line 27 columns to the right, which leaves the removed entry properly aligned over the right column.

3. If an item already exists in the LIFO stack, it simultaneously scrolls off-screen to make room for the newly removed item, which now occupies the top position in the stack.

Exceptions: There are three conditions under which the screen animation is different:

- **Removing the Top Item:** If the current item is the topmost item in the center column when the user presses the Remove Key, the remaining items in the column scroll *up* one screen line.

- **Leaving an Empty Column:** If the current item is the only item in the center column when the user presses the Remove Key, the columns move one column width to the right, leaving an empty right column with the parent of the removed item as the new current item. Whenever this occurs, the keyahead buffer is cleared, and no keyin is accepted until the cursor is turned back on marking the new current item.

- **Emptying the Hierarchy:** If the current item is the only item in the hierarchy when the user presses the Remove Key, the Remove operation leaves the center column set up for the user to enter a new current item from the keyboard. The inverse video bar does not move from its position, and the cursor appears in the first data column of the blank item.

Copy

The Copy operation allows the user to make an identical copy of the current item (and its hierarchical children) and place the copy at the top of the LIFO stack. The Copy operation is invoked by pressing and releasing the Copy Key — or, on the current general purpose keyboard, the F4 Key.

Screen Animation

When the user presses the Copy Key, the following screen animation occurs:

1. An identical copy of the current item appears in inverse video aligned over the center column on the Banner Line.

2. The program then smoothly scrolls columns 28 through 80 of the Banner Line 27 chracters to the right, which leaves the copied entry properly aligned over the right column. If an item already exists in the LIFO stack, it scrolls off-screen to make room for the copied item. The copied item is now the topmost item in the LIFO stack.

Recall

The Recall operation allows the user to recall the topmost item from the LIFO stack and insert it as the current item in the center column. The Recall operation is invoked by pressing and releasing the Recall Key — or, on the current general purpose keyboard, the F1 Key.

Screen Animation

The screen animation for the Recall operation is the exact opposite of the Remove animation. When the user presses the Recall Key, the following animation occurs:

1. Columns 28 through 80 on the Banner Line move 27 columns to the left, aligning the item being recalled over the center column.

2. Simultaneously, the next recallable item from the stack scrolls on screen to align with the right column.

3. Then, the item being recalled disappears from the Banner Line and appears as the current item in the center column. The current item and the items above it in the center column move up one screen line to make room for the recalled item.

Recall with the Insert Operations: If the user invokes one of the insert operations (Insert, InsertLeft, or InsertRight), then presses the Recall Key while the cursor is over the first data column of the new item, Recall results in moving the topmost item in the LIFO stack into the position opened up by the Insert operation.

Discard

The Discard operation allows the user to delete permanently the topmost item in the LIFO stack. The Discard operation is invoked through a ring menu. Miller column applications that support the Discard operation must use ''Discard'' as the ring menu option string through which it is invoked.

Screen Animation

When the user selects the Discard operation, the following screen animation occurs:

1. The first and last characters of the inverse video item that appears in the stack position on the Banner Line are replaced with normal video spaces.

2. Step 1 reiterates until the item has been completely erased.

3. The new topmost item in the stack moves in from off-screen until it is aligned over the right column.

NOTE: Once discarded, an item can never be recalled.

Rename

The Rename operation allows the user to change the text associated with the current item. The Rename operation is invoked through a ring menu. Miller column applications that support the Rename operation must use ''Rename'' as the ring menu option string through which it is invoked.

Screen Animation

When the user invokes the Rename operation, the cursor appears over the first text character of the current item (not over the padding space to its left). Entry of the new current option conforms to the non-destructive field keyin standards discussed in Chapter 5, ''The Field Keyin Technique.''

Merge

The Merge operation is the most complex of Miller column operations, allowing the user to remove the current entry while retaining its descendants. First the current entry is removed to the LIFO stack, then its descendants are promoted one hierarchical level (i.e., one column position to the left), filling and often expanding the gap left by their common ancestor. The Merge operation is invoked through a ring menu. Miller column applications that support the Merge operation must use ''Merge'' as the ring menu option string through which it is invoked.

Screen Animation

When the user invokes the Merge operation, the following screen animation occurs:

1. The current item is removed; the accompanying animation is the same as it would be if the user had pressed the Remove Key.

2. The items in the rightmost column move up until the bottommost item appears on the tenth screen line.

3. The bottommost item disappears from the right column and reappears as the current item in the center column. The items in the right column move down one line. The previous current item and any other items that appeared above it move up one screen line in the center column to accommodate the item that has moved over from the right column.

4. Step 3 reiterates until there are no items left in the right column.

Application-Dependent Extensions

This document has described a significant number of standards for the Miller column technique. However, these standards should not discourage application designers from implementing extensions that are appropriate for particular product requirements.

If you need to include any of the features we have discussed, you should do so in the manner described in this chapter. However, if you need to implement a feature we have not discussed, you need only ensure that your design and implementation are in keeping with the general principles and guidelines presented throughout this document. For example, VISTA-GUIDE displays a help message about the current item whenever you press the Help Key.

CHAPTER 4.
THE RING MENU TECHNIQUE

Contents

General Principles

Most programs allow users to specify options or commands in one form or another. RMS utilities support option specification through a combination of short text strings, commas, semicolons, and hyphens. IEOS provides its Command Line. RMS Mail and Multiplan support versions of ring menus that are functionally dissimilar.

Before the release of this document, there was no standard convention for command entry or option specification. The RMS Dashboard was designed to provide a standard method: the ring menu technique. All programs that require an option specification mode should use the ring menu technique, which consists of

- a screen format,
- definitions for 5 control keys, and
- description of some related screen behavior.

Ring menus are designed to work effectively in Miller column, freeform text, and field keyin applications. Some Miller column applications support the Discard, Rename, and Merge operations through a ring menu. Freeform text and field keyin applications can also use ring menus; for example, VISTA-SCRIPT™ supports field keyin within the Dashboard Frame and provides a ring menu on the Option/Prompt Line.

The Purpose of Ring Menus

A ring menu consists of a series of words or phrases, each of which corresponds to an activity you might want to perform in a given context. For example, here is one of the ring menus that the OOZL editor displays:

```
OOZL 1.1.j
 Find  Remember  Tabs  Window End Uppercase
```

"Find," "Remember," "Tabs," "Window," "End," and "Uppercase" are called *option strings*. Whenever a ring menu appears on the screen one of its option strings appears in inverse video. This is the *current option* and, like the current item in a Miller column display, is the user's focal point. The cursor appears immediately to the left of the current option. Using cursor movement keys, you can move the cursor (and the inverse video bar) from one option to another and select the one you want by pressing the Return Key.

At any time, a ring menu allows a user to do any of three things:

- invoke the current option,
- select a different current option, or
- exit the ring menu and resume the previous activity.

Occasionally ring menus are nested; that is, invoking the current option causes a new ring menu to appear so the user can provide a more detailed description of the desired activity. When ring menus are nested, an additional function should be supported: you must also be able to move from one level to the next.

This chapter discusses each of these functions in detail and provides specific implementation guidelines.

What a Ring Menu Looks Like

The Option/Prompt Line of the RMS Dashboard is reserved expressly for displaying a ring menu and any prompts that may go along with it.

Ring menus normally occupy a single screen line. There may be products in which this goal is impossible to attain. When no alternative is available, ring menus may occupy two lines (the Option/Prompt Line and the first line of the Help/Response Window). No ring menu, however, should ever occupy more than two screen lines.

Option Strings

A ring menu consists of a series of *option strings* that appear on the Option/Prompt Line of the RMS Dashboard. Each option string consists of a word or short phrase, the first letter of which is capitalized.

Option String Padding Spaces

A blank space always appears on each side of each option string in a ring menu. These blanks are called *padding spaces*. When an option string appears in inverse video, so do its padding spaces.

Bringing Up a Ring Menu

A ring menu appears only when a user specifically requests it by pressing the Command Key. When the ring menu first appears, the cursor appears one column to the left of the left-hand padding space of the first option string.

The Current Option

One option string is always the current option. This option appears in inverse video (along with its padding spaces). The cursor always appears one position to the left of the left-hand padding space of the current option string. For example:

```
Incoming  Outgoing  Lookup  Utility  Exit
```

Spacing Between Option Strings

The Option strings are spaced as evenly as possible across the screen. The left-hand padding space of the leftmost option string appears in column 1 on screen line 20. At least one but not more than five spaces (excluding padding spaces) separate each option string.

If the only alternative is to display a ring menu on two lines rather than one, you may omit these separating spaces, resulting in a series of option strings that are separated only by their padding spaces. For example:

```
Incoming  Outgoing  Lookup  Utility  Exit
```

Ring Menu Operations

Option specification is a simple matter. Ring menus recognize only a small number of keys:

- Command Key
- Quit Key
- Left Arrow Key
- Right Arrow Key
- Return Key.

This section describes the behavior of these keys. If you want to read the general guidelines that apply to these keys, please see Chapter 2, ''General Standards.''

Mode Switching

Entering and leaving a ring menu involves moving the user back and forth between two hard modes. The Command, Quit, and Return Keys support the required mode switching. These keys are discussed in more detail below.

The Command Key

The Command Key functions as a toggle switch between normal operation mode and option specification mode. You use the Command Key to enter and exit a ring menu.

When you press the Command Key, a ring menu appears. If you press it again, the ring menu vanishes, and the focus of the user's attention returns to the context from which the ring menu was entered.

When a ring menu first appears, the leftmost option string on the Option/Prompt Line is the current option.

The Quit Key

The Quit Key enables users to leave a context — for example to exit SCOUT or to return to VISTA-GUIDE from an activity script without running it.

In a ring menu, the Quit Key allows users to leave the ring menu and return to the interactive context of the current program. When you press the Quit Key, the ring menu disappears from the screen, and the cursor returns to the location it occupied at the time you requested the ring menu.

The Return Key

The Return Key lets users select a command or option. Pressing the Return Key instructs a program to do the activity specified by the current option. This often entails changing or somehow manipulating the data or objects in the Dashboard Frame.

Pressing the Return Key causes the ring menu to vanish and generally causes the cursor to return to its former position within the Frame. Sometimes, however, when a program needs to collect additional information from a user about the option he has selected, the ring menu is replaced by one or more prompts on the Option/Prompt Line. Depending on the nature of the information to be collected by a prompt, programs should utilize the standard field keyin or freeform text model to collect additional information.

Once in a while, a ring menu is replaced by another ring menu. This situation is discussed completely under ''Nested Ring Menus'' later in this chapter.

Moving Around in a Ring Menu

There are two methods for changing the current option, which are described below.

The Left and Right Arrow Keys

Ring menus support the Left and Right Arrow Keys for cursor movement. The Left Arrow Key moves the cursor (and the inverse video bar) one option string to the left. The Right Arrow Key moves them one option string to the right. Both keys support wrapping at the first and last option string. This is what makes them *ring* menus.

Alternative Cursor Movement

When a ring menu includes numerous options, it can be cumbersome to use the Left and Right Arrow Keys to move to the option you want. The technique permits users to minimize keystrokes and to keep their hands in home position by using an alternative method of cursor movement.

Each option string begins with a capital letter. You can move the cursor immediately to an option by typing its capitalized letter.

```
  Incoming  Outgoing  Lookup  Utility  Exit
```

In this ring menu you can type a *u* to select ''Utility'' as the current option.

As a general rule, interactive software should avoid using option strings that begin with the same letter. If, however, the only alternative is to use an option string that inadequately suggests the function it corresponds to, option strings may commence with the same letter.

When two or more option strings begin with the same letter, typing that letter causes the cursor (and the inverse video bar) to advance to whichever option begins with that letter and would be reached first by pressing the Right Arrow Key. For example:

```
Enter  Igloo  Exit  Bird  Fill  End  Fickle
```

Because of the current cursor position, typing an *e* selects "End" as the current option. Typing another *e* selects "Enter."

Nested Ring Menus

From time to time it may be desirable to nest ring menus. For example, VISTA-SCRIPT allows you to save an activity script by invoking the Save option on the ring menu:

```
Run Save Add to chain  Quit
```

You select the Save option by typing an *s* and pressing the Return Key. Because there are three different ways to save an activity script, VISTA-SCRIPT now displays another ring menu:

```
Update script Save new script  save Command
```

This ring menu is nested beneath the Save option on the first-level VISTA-SCRIPT ring menu. The user only confronts the variations on the Save operation when he is actually in the process of saving an activity script.

When necessary, user programs may implement nested ring menus. However, there should never be more than two levels of ring menus.

In a nested ring menu, the Enter and Command Keys behave as described earlier. The Quit Key, however, operates somewhat differently in a second-level ring menu. When a user presses the Quit Key, the program returns him to the first-level ring menu, with the cursor (and inverse video) positioned over the previously current option (i.e., the option that expanded into the second-level ring menu).

CHAPTER 5.
THE FIELD KEYIN TECHNIQUE

Contents

General Principles

The field keyin technique allows programs to collect data strings from a user. Field keyin is quite common in numerous existing programs. It allows the VISTA-SCRIPT component of the RMS command interpreter to collect information about how a user wants to perform a particular task. In more complex situations, field keyin is but a small part of a larger context, as in VISTA-SCRIPT, where collecting the label of an item is accomplished through the standard field keyin mechanisms described in this chapter.

Unlike Miller columns and ring menus, field keyin has no particular screen format associated with it. It fits appropriately into the Dashboard Frame, the Option/Prompt Line, or the Help/Response Window, as required by the application.

The field keyin technique described in this chapter is properly implemented in the VISTA-SCRIPT component of the RMS 2 command interpreter. If you would like to see an example of the technique, simply run VISTA-SCRIPT.

Field Keyin Is Non-Destructive

The DATAPOINT field keyin technique lets you enter data into a field without committing yourself to it until you enter a confirming keystroke. At any time, you can recall data that was previously ``stored'' in the field without retyping (or even remembering) it. We talk about this technique as ``non-destructive field keyin.''

These non-destructive keyin techniques are not supported by the RMS nucleus. Rather, programs that need to support field keyin must create their own field keyin logic using standard RMS single character keyin mechanisms. The techniques, however, are extremely simple.

Initial Field Display

If data for a particular field already exists, the program displays this data intact in the field. When the user selects that field for editing or entry, the program places the cursor over the first character of the data in that field.

Field Editing

As soon as the user enters the first new data character, the pre-existing data vanishes, and the program begins to echo the user's input. Programs must screen input intelligently, accepting data that is alphanumeric or numeric only, depending upon the requirements of the program. Workstations do not beep when users enter invalid data during field keyin; rather, the program simply disregards and does not echo invalid keystrokes.

Additional Keys

Programs supporting field keyin acknowledge four special keys in addition to those they accept as user data input.

Backspace: The Backspace Key behaves normally, except that when the user backspaces to the beginning of the field, any data which was there before modification began reappears in the field.

Undo: Pressing the Undo Key has the same effect as backspacing to the beginning of a field. If the user presses the Undo Key at any time prior to pressing the Return Key, the program cancels any modifications that have been made and redisplays any data which was previously in the field, placing the cursor again over the first data character.

Enter: The Return Key confirms the data that currently appears in the field. If the user has entered new data, the Return Key confirms that; otherwise, field keyin is terminated, retaining the original data.

Quit: Pressing the Quit Key at any time causes field keyin to terminate with the original value intact and correctly displayed.

CHAPTER 6. THE FREEFORM TEXT EDITING TECHNIQUE

Contents

General Principles

The freeform text editing technique allows programs to collect arbitrary streams of characters from a user. Programs supporting freeform text entry or editing must process each user keystroke as a single entity.

Freeform text editing forms the basis of word processing programs (like IEOS and WP) and text editors (like OOZL and EASL). There is a particular screen format associated with the freeform text editing technique (p 2-13). It includes definitions of

- eight cursor movement keys,
- two scrolling keys, and
- four editing keys.

Freeform text editing generally occurs in one of two types of editors:

Line Editors: As the name suggests, line editors deal with single lines, often called records, of text. Line editors generally do not provide word wrap or other WP-type features, although some do provide limited facilities in this area.

Stream Editors: Stream editors deal with streams of text that conform to their run-time containers. Stream editors always support word wrap, although users can often override it. Most word processing systems are stream editors.

Both line and stream editors should always place the user in insert mode as the default. An editor may also support typeover mode; but, insert mode should be the default. A method of implementing typeover mode is presented in the discussion of the Insert Key under ''Editing Keys'' at the end of this chapter.

Moving Around in Freeform Text

The cursor movement keys allow the user to change his position within freeform text.

Up and Down Arrow Keys

The Up and Down Arrow Keys move the cursor a single screen line up or down respectively. These keys cause scrolling whenever necessary to keep the cursor on the screen.

Left and Right Arrow Keys

The Left and Right Arrow Keys move the cursor a single column to the left or right respectively. These keys cause horizontal scrolling whenever necessary to keep the cursor on the screen.

Stream Editor: In a stream editor, the Left and Right Arrow Keys wrap at line boundaries.

Line Editor: In a line editor, the Left and Right Arrow Keys do not wrap at line boundaries.

Bottom Left and Bottom Right Corner Keys

The Bottom Left and Right Corner Keys move the cursor backward and forward by word, always leaving the cursor over the first character of a word. These keys also cause scrolling whenever necessary to keep the cursor on the screen.

Stream Editor: In a stream editor, these keys wrap at line boundaries.

Line Editor: In a line editor, the BLC or BRC Key must be pressed twice to accomplish wrapping. For example, if you press the BRC Key while the cursor is within the final word in a record, the cursor advances to one column to the right of the last nonblank character in the record; if you press it again, the cursor moves to the first nonblank character in the next record.

Top Left and Top Right Corner Keys

The Top Left and Right Corner Keys move the cursor backward and forward by whatever unit is logically next larger than a word. In a stream editor, this is generally a paragraph; in a line editor, it is generally a record. These keys also cause scrolling whenever necessary to keep the cursor on the screen.

Line Editor: In a line editor, the TLC Key moves the cursor to the first nonblank character in the current record; the TRC Key moves the cursor to one column to the right of the last nonblank character in the current record. These keys do not wrap at record boundaries.

Stream Editor: In a stream editor, the TLC and TRC Keys generally cause cursor movement by paragraph. The TLC Key moves the cursor to the first nonblank character in the current paragraph or, if the cursor is already there, the first nonblank character in the preceding paragraph. The TRC Key moves the cursor to the first nonblank character in the next paragraph.

Window Key

In freeform text editing applications, the Window Key performs no independent function but rather serves as a toggle switch between two modes: one for normal editing operations, the other for *express cursor movement*.

When a user presses the Window Key, he enters express cursor movement mode. The next keystroke completes the express cursor movement. It may be processed in several different ways; each possibility is discussed below.

Non-Cursor Movement Key

If the user presses the Window Key followed by any key other than one of the cursor movement keys, express cursor movement is terminated, the normal system cursor reappears on the screen, and normal editing resumes.

Left or Right Arrow Key

If the user presses the Window Key followed by the Left Arrow Key, the cursor moves to the left margin or left screen boundary on the current screen line, whichever is closer to the current cursor position.

The animation becomes more complicated when the editing container is wider than the physical workstation screen. If the cursor is at the screen boundary and the left margin is still off-screen, the key sequence moves the cursor to the left margin, scrolling as necessary to keep the cursor on-screen. This means users may have to press WindowLeft twice — once to get to the screen boundary and once to get to the left margin.

The Right Arrow Key works the same way but in the opposite direction.

Up or Down Arrow Key

If the user presses the Window Key followed by the Up Arrow Key, the cursor moves to the top of the text container or the top screen boundary in the current column, whichever is closer to the current cursor position.

If the cursor is at the screen boundary and the top of the container is still off-screen, the key sequence causes the text to move downward such that the previous top line becomes the new bottom line. The cursor remains on the top screen line. If there is not enough document to do this, the first line of the document becomes the new top line.

The Down Arrow Key works the same way, but in the opposite direction.

Corner Keys

If the user presses the Window Key followed by the Top Left Corner Key, the cursor moves to the intersection of the left margin and the top of the editing container or the top screen boundary, whichever is closer to the current cursor position.

If the cursor is at the top left screen boundary and the top of the container or the left margin is still off-screen, the key sequence causes the text to move downward such that the previous top line becomes the new bottom line and the cursor moves to the left margin. If there is not enough document to do this, the first line of the document becomes the new top line. The cursor appears at the intersection of the left margin and the top screen line.

The other corner keys cause the same series of actions in the appropriate directions.

The Scrolling Technique

Scrolling occurs whenever a user attempts to move the cursor outside of the window through which he is currently viewing some element of the system. For example, pressing the Down Arrow Key eventually moves the cursor onto the last screen line; pressing it again causes the text to move up a line so that the cursor remains visible on the screen.

From time to time, it is desirable to cause scrolling without moving the cursor. The DSP and KBD Keys on the current general purpose keyboard support scrolling. The DSP Key, also referred to as the Downscroll Key, moves the user's perspective down and therefore the text up. The KBD Key, also referred to as the Upscroll Key, moves the user's perspective up and therefore the text down.

On the RMS 2 keyboard, scrolling occurs when a user chords either Alt Key with any of the arrow keys. For example, AltUp scrolls the user's perspective up and therefore the text down. AltLeft scrolls the user's perspective left and therefore the text right.

Editing Keys

Chapter 3, ''The Miller Column Technique,'' describes a recallable LIFO stack available in applications that support editing a Miller column hierarchy. This same LIFO stack is important to effective text editors.

In a Miller column application the nature of the item you are manipulating is always clear; it is a 23-character text string that appears in the center column. In freeform text editing, however, users need to define the size and shape of the item they wish to manipulate. This is accomplished by a block selection technique based on chording editing keys with cursor movement keys to produce the effects described below.

Remove Key: The Remove Key removes an item from the text and places it in the LIFO stack. Upon receiving the downstroke of the Remove Key, the program disables all input other than cursor movement keys. As the user presses cursor movement keys, the cursor advances and the text that it passes is placed in inverse video. When the user releases the Remove Key, the selected text is removed to the LIFO stack.

The Copy Key: The Copy Key duplicates an item from the text to the LIFO stack. Upon receiving the downstroke of the Copy Key, the program disables all input other than cursor movement keys. As the user presses cursor movement keys, the cursor advances and the text that it passes is placed in inverse video. When the user releases the Copy Key, the selected text is copied to the LIFO stack.

The Recall Key: The Recall Key recalls the top-most item in the stack, inserting it into the text starting at the current cursor position. The length of the item is determined by the operation that originally placed it in the stack. For example, the topmost item in the stack may be a page, the next item a paragraph, and the next a few words.

The Insert Key: The Insert Key functions as a toggle switch between insert and typeover modes in those editors that support a typeover mode. Whenever the user is in typeover mode, the program displays the message "TYPEOVER" in the Help/Response Window.

NOTES

CHAPTER 7. TECHNICAL CONSIDERATIONS AND GUIDELINES

Contents

Program Performance

Program performance plays a significant role in determining the ‘‘feel’’ of a product, its potential success in the marketplace, and the degree to which customers find it useful. A certain level of program performance is critical to properly interactive software, and this document specifies minimum program performance standards.

A New Context for Software Development

The introduction of VISTA-VIEW, ATTACH, and CPF imposes significant WSIO performance overhead. For example, consider a program that reads a single character and echoes it on the workstation screen. Under the current system architecture, the character follows this path:

1. Keyboard
2. Nucleus
3. User program
4. Nucleus
5. Screen

When VISTA-VIEW is running, however, the character must follow a far more complex path:

1. Keyboard
2. VISTA-VIEW task's nucleus
3. VISTA-VIEW
4. VISTA-VIEW task's nucleus
5. Pipe
6. VISTA-VIEW channel's nucleus (potentially local or remote)
7. Second PCR
8. User program
9. VISTA-VIEW channel's nucleus
10. Second PCR

11. VISTA-VIEW channel's nucleus
12. Pipe
13. VISTA-VIEW task's nucleus
14. VISTA-VIEW
15. VISTA-VIEW task's nucleus
16. Screen

Eventually, most users may access all software products through VISTA-VIEW. When the keyin/echo/ready operation constitutes the primary activity of a program — as it does in WP, RMS Mail, and other highly interactive programs — and performance is additionally encumbered by redirected workstation I/O, serious performance optimization becomes essential.

VISTA-VIEW, ATTACH, and CPF are being optimized, but application developers must create optimized programs on the user level as well.

The Objective

DATAPOINT has established a minimum performance requirement for all RMS-targeted interactive software: The keyin/echo/ready loop may take no longer than 60 milliseconds running in a local processor under VISTA-VIEW. The following guidelines should assist you in achieving this objective.

Minimize WSIO System Calls: To the greatest extent possible, reduce the number of WSIO calls required to complete any given operation. There is a significant amount of overhead associated with each call. Each WSIO call should perform the greatest amount of work possible. In any event, programs processing field keyin should issue no more than one WSIO call per field, and programs processing character keyin should issue no more than one WSIO call per user keystroke, including the updating of counters, ruler lines, and similar screen elements.

Use $ESNF Instead of $ES: RMS 2 supports a new WSIO control code that simplifies the packaging of WSIO calls. Strings terminated with a $ESNF are buffered until a $ES is encountered. The $ESNF control code can provide measurable performance benefits.

Sensible Screen I/O: Under DATAPOINT's traditional architecture, we have been able to rewrite vast amounts of the screen with minimal performance penalties. When workstation I/O is redirected, though, the penalties become more pronounced. Programs must avoid redisplaying information that already appears on the screen, instead saving more screen state information internally and updating the screen selectively. Additionally, programs should use an ERASE EOF or ERASE EOL rather than writing a large number of blank characters.

Use the Full WSIO Facility: The WSIO facility supports more powerful screen manipulation than many programs use. The WP-style insert facility, for example, saves an average of one thousand characters sent to the screen each time the Insert Key is pressed. This facility was added to WSIO to emulate exactly the capability required by the WP editor. Failure to use such facilities effectively eliminates multi-user operation of DATAPOINT processors in the office environment — even without VISTA-VIEW, ATTACH, or CPF.

International Transportability: The Standard Workstation Font

All RMS software must be internationally transportable. Historically, the major obstacle to transportability has been the lack of agreement on the character set. A standard workstation font has now been designed. Programs may no longer load their own character sets; rather, they must use the standard character set that appears on the next page.

Special Symbols: Domestically, characters in the range 0002-0032 support certain special symbols, and 0033-0037 are not used. In other countries, however, characters in the range 0002-0037 may be national data characters, and the special symbols may be unavailable. This reality imposes certain constraints on programs that use characters in the range 0002-0037:

- Characters in the range 0002-0037 must be message member translatable;
- The absence of these characters may not damage the program's functionality or usefulness. (Since these characters are not available in all countries, no program whose functionality depends upon them is internationally transportable.); and
- These characters may not be used in program control sequences. For example, no application may support a key chord like RemoveA because some countries do not have A in their alphabet.

Programs must, of course, support message member translatability for error messages, prompts, and other language-dependent program mechanisms.

THE STANDARD WORKSTATION FONT

0000	System cursor
0001	VISTA-VIEW quiescent cursor

BOX-DRAWING CHARACTERS

0002	Upper left corner
0003	Upper right corner
0004	Lower left corner
0005	Lower right corner
0006	Vertical bar
0007	Horizontal bar
0010	Upper vertical divider
0011	Lower vertical divider
0012	Left horizontal divider
0013	Right horizontal divider
0014	Center divider

KEYBOARD SYMBOLS

0015	Up arrow
0016	Down arrow
0017	Left arrow
0020	Right arrow
0021	Enter Key
0022	Command Key

VISTA-VIEW CURRENT WINDOW SYMBOLS

0023	Upper left corner
0024	Upper right corner
0025	Lower left corner
0026	Lower right corner
0027	Horizontal top bar
0030	Horizontal bottom bar
0031	Vertical left bar
0032	Vertical right bar

0033 thru 0037
Unused

0040 thru 0176
ASCII 7-bit printable character set

0177 Meta character