TEK PROGRAMMER'S REFERENCE

Part No. 070-4172-00
Product Group 11

4025A
COMPUTER DISPLAY TERMINAL

Please Check for
CHANGE INFORMATION
at the Rear of this Manual

First Printing OCT 1981
Revised DEC 1981

Tektronix®
COMMITTED TO EXCELLENCE

WARNING

This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the instruction manual, may cause interference to radio communications. It has been tested and found to comply with the limits for Class A computing devices pursuant to Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference when operated in a commercial environment. Operation of this equipment in a residential area is likely to cause interference in which case the users at their own expense will be required to take whatever measures may be required to correct the interference.

This instrument, in whole or in part, may be protected by one or more U.S. or foreign patents or patent applications. Information provided on request by Tektronix, Inc., P.O. Box 500, Beaverton, Oregon 97077.

MANUAL REVISION STATUS

PRODUCT: 4025A Computer Display Terminal

This manual supports the following versions of this product: Firmware Version 3.0 and up.

REV.	DATE	DESCRIPTION
@	OCT 1981	Original Issue
-	DEC 1981	Revised: page D-1.

CONTENTS

Section 1 **INTRODUCTION** **Page**

Section 2 **COMMAND STRUCTURE**

Section 3 **HOST PROGRAMMING**

Section 4

PROGRAMMING THE KEYBOARD

Section 5

SYSTEM STATUS AND INITIALIZATION

Section 7 (cont)

Section 8

TEXT EDITING

Section 9

GRAPHICS

ILLUSTRATIONS

TABLES

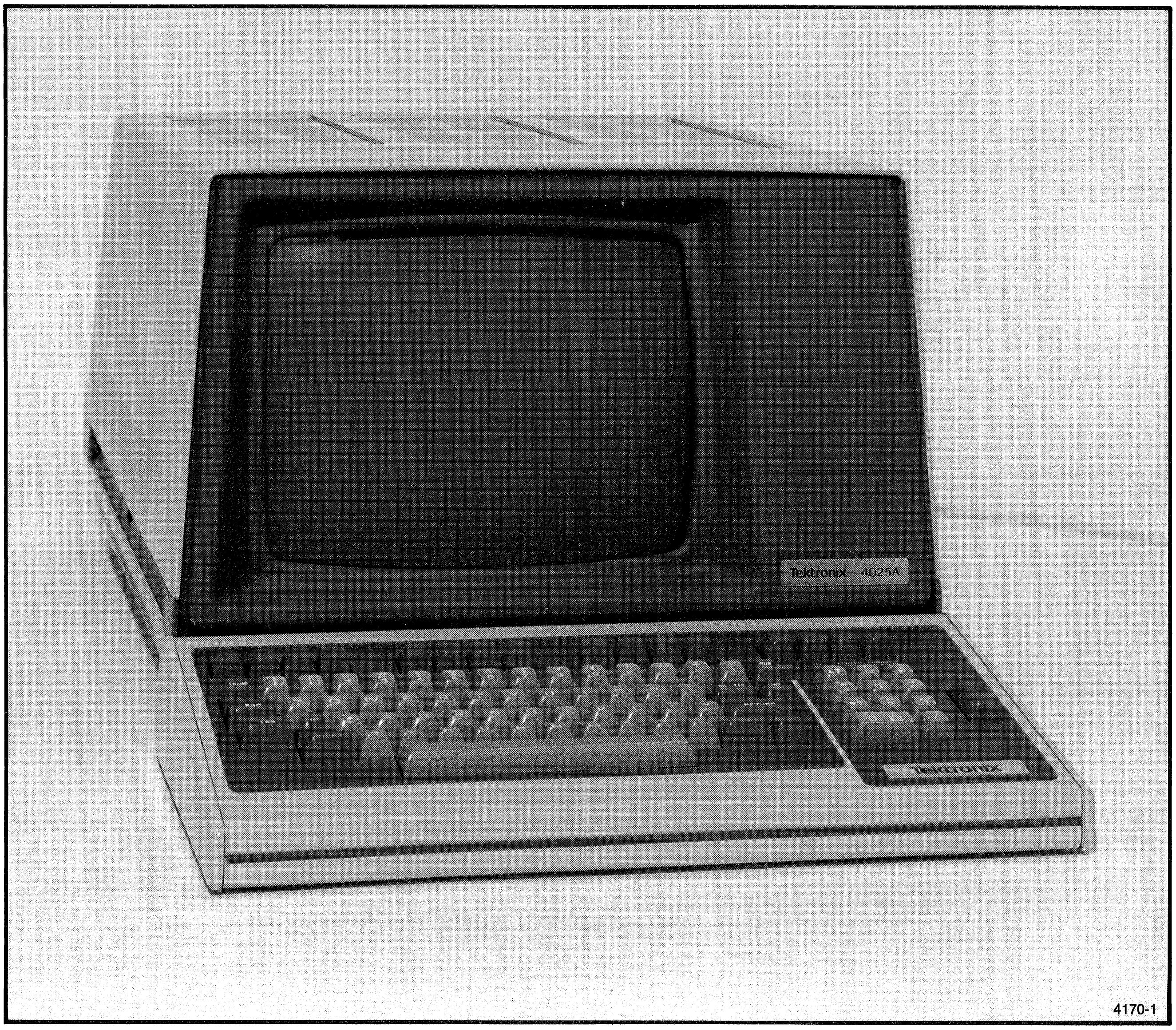


Figure 1-1. The 4025A Computer Display Terminal.

Section 1

INTRODUCTION

The 4025A Computer Display Terminal is what is popularly known as a "smart terminal." The terminal carries communications between the operator and a host computer. In addition, the terminal contains its own microprocessor and supporting electronics. For this reason, the terminal also responds to its own set of commands, independently of the host computer.

The terminal is not intended to be a stand-alone computing system. Its computing ability complements that of the host computer to enable the user to make full use of the terminal's information display capabilities.

ABOUT THIS MANUAL

This manual describes in detail the commands to which the terminal responds. The user of this manual should be familiar with computer operations in general and with at least one programming language. The user should also have access to the 4025A Operator's Manual.

The 4025 Programmer's Reference Manual is organized along broad functional lines. Section 1 gives an overview of the 4024/4025 Computer Display Terminal. Each succeeding section explores one class of commands related to a basic terminal function.

This section is an overview of the 4025A Computer Display Terminal. Each succeeding section describes one class of commands related to a basic terminal function. The terminal has a variety of parameter settings, most of which are set by command, and the action of other commands may be influenced by these settings. When this is the case, commands are cross-referenced. An appendix contains a list of commands and their flags in summary form.

RELATED DOCUMENTATION

Related programming information can be found in the following documentation:

4025A Computer Display Terminal Operator's Manual

4010BO1 (B02,B03,B04,B05) Plot 10 Easy Graphing Software documentation

A 4025A Programmer's Reference card, containing a summary of information in this manual, is also available.

THE 4025A COMPUTER DISPLAY TERMINAL

The 4025A Computer Display Terminal (Figure 1-1) is an interface between the terminal operator and a host computer. It is designed especially for applications involving text editing and display and processing of forms. The terminal also possesses basic graphic capability.

The terminal consists of a display unit and a keyboard attached to the display unit by a thin flexible cable. The display unit contains a 12 inch, refresh-style crt, a microprocessor with supporting electronics, and a standard RS-232 interface. The terminal operator types information on the keyboard. Information from both the keyboard and the host computer is displayed on the crt.

Terminal operations are controlled by the microprocessor and its associated firmware (programs for the microprocessor which are stored in Read Only Memory chips, or ROMs). With this firmware, the terminal responds to several dozen commands, independently of the host computer. These commands determine settings of the terminal system parameters, control the screen display, and perform various functions useful in applications programs.

Terminal Features

- Workspace and Monitor — Display memory can be divided into two portions or scrolls. One portion, called the workspace, serves as a scratchpad for editing text, filling out forms, or displaying results of applications programs. The monitor portion of memory stores messages to and from the computer and any commands typed on the keyboard.
- Split Screen — The screen can be divided into two areas or windows, corresponding to the two portions of the display memory. The upper area is the workspace window and displays information from the workspace. The lower area is the monitor window and displays information from the monitor without writing over the workspace display. The portions of the screen allotted to each of these windows are set by command.
- Scrolling — When either the workspace window or the monitor window is full, information in that window scrolls up to display additional information. Infomation scrolled off the screen is saved as long a memory is available; the scrolled text may be reviewed by scrolling down.
- Forms — The workspace can display a form. When the operator has filled in the blanks of the form, the data in these blanks can be sent to the computer with a single command.

- Visual Enhancements — Characters can be displayed with the standard visual attribute (light characters on dark background) or the enhanced visual attribute (brighter characters on light background). The display can be blinked between these attributes. The terminal can also display visual attributes of underscored (all characters and spaces underlined), inverted (dark characters on light background), and combinations of more than one visual attribute (such as enhanced with underscore). Screen brightness and contrast are controlled manually by the operator.

- Locally or Remotely Programmable — Commands to the terminal can be typed on the keyboard or sent from the computer.

- Programmable Operating Parameters — Various operating parameters (such as parity, workspace margins, tab stops, etc.) can be set by commands given either from the keyboard or from the computer.

- Buffered Operation — In buffered mode, a line of text (up to 80 characters) in the monitor is saved for proofing or local editing before it is sent to the computer.

- Programmable Keyboard — Almost all of the keys on the keyboard can be programmed to generate a different character or character string than the default one. This allows commonly used character strings or commands to be generated by pressing a single key.

- Local Text Editing — Using the editing keys or commands, one can edit text held in the workspace before sending it to the computer.

- Status Messages — The terminal can display status messages which indicate parameter settings, the command character, and the amount of unused memory in the terminal.

- Graphics — The terminal can store and display graphs in the workspace. Solid lines and several types of dashed lines can be drawn.

- Programmable Baud Rate — The terminal baud rate can be set by command.

- Hard Copies — The 4025A can make permanent copies of all information on the screen using a TEKTRONIX 4611 or 4631 Hard Copy Unit. The Hard Copy Unit copies forms and graphs just as they appear on the 4025A screen.

Optional Features

- Printer copies — Information in the workspace or in the computer can be copied on a TEKTRONIX 4642 Printer.
- Rulings — The Ruling Characters font provides a variety of ruling characters. Using this font, the terminal can draw horizontal and vertical rulings to highlight the structure of a form displayed in the workspace.
- GIN — GIN (Graphics Input) mode provides a means to digitize and send graphics information to the host computer.
- Additional Display Memory — Options provide up to 32 K bytes of display memory.
- Optional Interfaces — Options allow the terminal to use a 20 mA current loop or an RS-232 peripheral communications line.
- Alternate Character Fonts — The Math Characters font provides a variety of symbols useful in mathematical applications. The user can define 16 character fonts, with each font containing up to 128 characters.
- GPIB Interface — The terminal can communicate with four TEKTRONIX 4924 Digital Cartridge Tape Drives and two TEKTRONIX 4662 Interactive Digital Plotters, using a GPIB (General Purpose Interface Bus).

THE SPLIT SCREEN: WORKSPACE AND MONITOR

Information sent to the display unit from the keyboard or the computer is stored in a part of the terminal's memory called the display list. This display list can be divided into two sections or scrolls — the workspace scroll (or simply workspace) and the monitor scroll (or simply monitor).

Information from the keyboard can be directed into either scroll, as can information from the computer. Each scroll has specific uses, and the terminal processes information in the workspace differently than it processes information in the monitor.

The workspace serves as a scratchpad area. The operator can use it to create text to send to the computer, to edit text, to create or fill out forms, or to display results of applications programs. Text typed into the workspace is stored there until the terminal is commanded to send data in the workspace to the computer.

The monitor is used to display commands typed on the keyboard and messages to and from the computer. The monitor cannot contain forms or graphics. In general, the monitor allows (1) the operator to communicate with the terminal or the computer, and (2) the computer to issue error messages or prompts, without this information being written over the contents of the workspace.

Corresponding to this division of the display list into workspace and monitor, the terminal screen can be divided into two areas or "windows" to display information stored in each part of the display list. The upper area is the workspace window and displays information from the workspace. The lower area is the monitor window and displays information from the monitor.

There is always a monitor defined; hence there is always a monitor window of at least one line. There may, however, be no workspace defined. If no workspace is defined, there is no workspace window; the entire screen is devoted to the monitor.

When the terminal is powered up or RESET, the monitor window occupies the entire 34 lines of display, no workspace is defined, and text from the keyboard and text from the computer are directed into the monitor. Appropriate commands to the terminal define a workspace, select the number of lines in each window, and direct text from the keyboard and text from the computer into the desired scrolls.

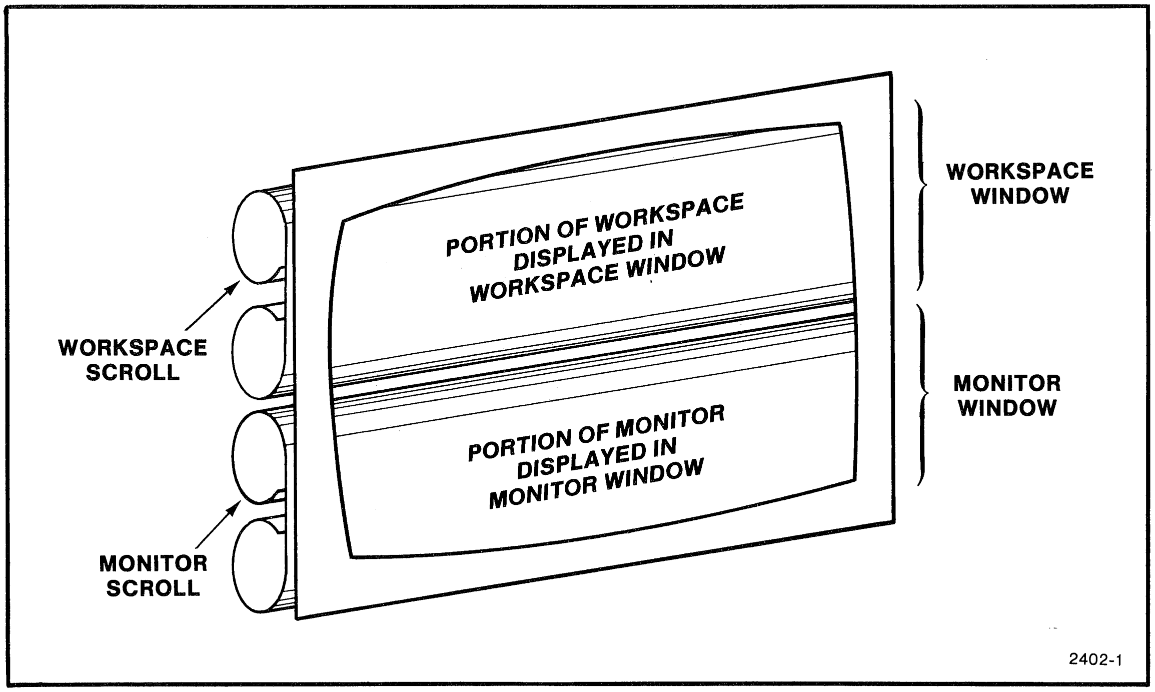

PORTION OF WORKSPACE DISPLAYED IN WORKSPACE WINDOW
PORTION OF MONITOR DISPLAYED IN MONITOR WINDOW
WORKSPACE WINDOW
MONITOR WINDOW
WORKSPACE SCROLL
MONITOR SCROLL
2402-1

Figure 1-2. The Split Screen; Workspace and Monitor Scrolls.

For each scroll there is a **cursor** — a pointer in the display list indicating where the next character entered in the scroll will be stored. The cursor appears on the screen as a bright underline one column in width. Only one cursor will be visible at a given time. (There may be brief periods, while the terminal performs certain routines, when neither cursor is visible.)

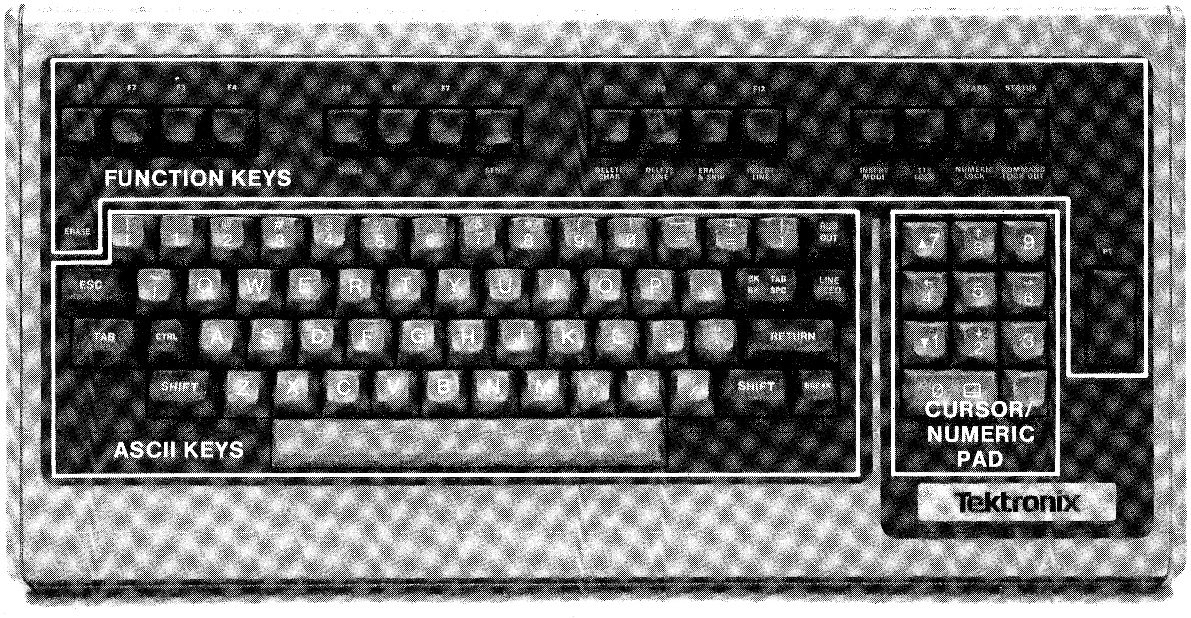
If the workspace window is full and additional text is entered in the workspace, the workspace automatically scrolls up to display the new text. Text scrolled off the screen is saved in the display list so long as that memory capacity is not exceeded. The operation of the monitor is similar, except that information scrolled off the monitor window will be discarded if that memory space is needed for other purposes.

Scrolling commands and scrolling keys roll the workspace and monitor up and down, independently, to display various portions of text.

THE KEYBOARD

The keyboard is shown in Figure 1-3.

As indicated in Figure 1-3, the keyboard can be divided into ASCII keys, cursor/numeric pad keys, and function keys.

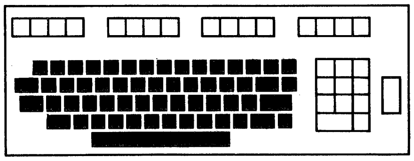


2657-1

Figure 1-3. The 4025A Keyboard.

ASCII Keys

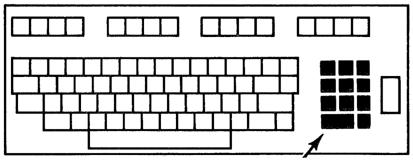
The ASCII section of the keyboard resembles an ordinary typewriter keyboard. Each key in this section, except the BREAK key, sends a character of the ASCII code to the computer. (See the ASCII Code Chart, Appendix B.) The BREAK key sends a break signal which interrupts the computer's operation.



ASCII KEYS

Cursor/Numeric Pad Keys

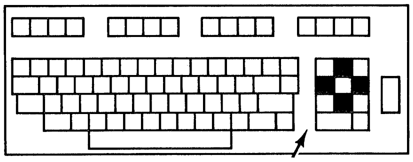
The cursor/numeric pad is the group of 11 keys to the right of the ASCII section of the keyboard. This group of keys functions as a cursor pad and as a numeric pad.



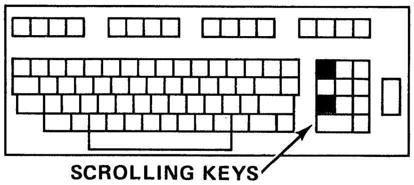
CURSOR/NUMERIC PAD

When the NUMERIC LOCK function key is off (unlighted), the group functions as a cursor pad. In this mode the four keys marked with arrows move the cursor and the two keys marked with triangles scroll the display list. The remaining pad keys have no effect.

If Option 25 or 26 and Option 38 are installed, these "arrow" keys move the crosshairs in the graphics display workspace.

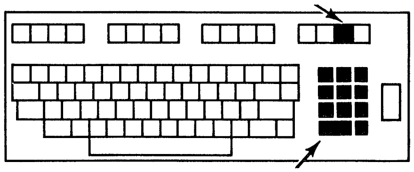


CURSOR MOVEMENT KEYS



SCROLLING KEYS

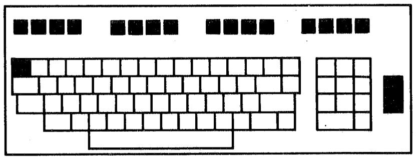
When the NUMERIC LOCK function key is on (lighted), the group functions as a numeric pad, generating the digits 0 — 9 and the decimal point (period). The shifted versions of the appropriate pad keys still move the cursor and scroll the display list.



NUMERIC LOCK

NUMERIC KEYS

Function Keys



FUNCTION KEYS

The function key group consists of the ERASE key, the PT (Pad Terminator) key, and the sixteen keys along the top of the keyboard.

The ERASE key is at the extreme upper left of the ASCII section of the keyboard. This key erases whichever scroll (workspace or monitor) receives text from the keyboard.

The PT (Pad Terminator) key is the large key to the right of the cursor/numeric pad. The default definition of this key is "undefined."

The sixteen keys along the top of the keyboard are divided into four groups of four each. Each key in the rightmost group includes an LED which, when lighted, indicates the key is "on." These sixteen keys have the following definitions.

F1 — Undefined
F2 — Undefined
F3 — Undefined
F4 — Undefined
F5 — HOME
F6 — Undefined
F7 — Undefined
F8 — SEND **
F9 — DELETE CHARACTER
F10 — DELETE LINE
F11 — ERASE & SKIP
F12 — INSERT LINE
F13 — INSERT MODE *
F14 — TTY LOCK *
F15 — NUMERIC LOCK/LEARN *
F16 — COMMAND LOCKOUT/STATUS *

*lighted keys
**The SEND key has no definition until programmed.

Function keys F1-F4 and F6-F8 have default definitions of "undefined"; these keys cause no action unless they are programmed.

Function key F5 is the HOME key. Pressing this key returns the visible cursor to its "home" position in row 1, column 1 of its scroll at the upper left corner of the window.

Function keys F9-F16 perform the functions indicated by their keyboard labels. These keys are discussed in detail in th Operator's Manual.

The default definition of the SEND key is "undefined." Since the command set includes two different types of SEND commands, the shifted and unshifted versions of the SEND key may be programmed, each with a different type of SEND command.

The LEARN key is the shifted version of the NUMERIC LOCK key. The STATUS key is the shifted version of the COMMAND LOCKOUT key. Neither the LEARN nor the STATUS key is a lighted key; each operates independently of the corresponding unshifted key.

The action of the DELETE CHAR, DELETE LINE, INSERT LINE, INSERT MODE, and LEARN keys can be duplicated by commands discussed later in this manual.

There are no commands that exactly duplicate the action of the HOME, ERASE & SKIP, TTY LOCK, NUMERIC LOCK, COMMAND LOCKOUT, or STATUS keys. The action of the HOME, ERASE & SKIP, and COMMAND LOCKOUT keys can be duplicated by certain command sequences discussed in later sections of this manual. There are no command sequences which duplicate the action of the TTY LOCK or NUMERIC LOCK keys.

Programmable Keyboard

Most of the keys on the keyboard can be programmed with definitions other than the default ones. This allows the operator to generate commonly used character strings, commands, or command sequences by pressing a single key.

All of the keys on the keyboard can be programmed except the following.

- The rightmost three lighted function keys — TTY LOCK, NUMERIC LOCK, and COMMAND LOCKOUT. (Neither the shifted nor the unshifted versions of these keys can be programmed.)
- The three ASCII keys — SHIFT, CTRL, and BREAK.

Key programming may assign different definitions to the shifted and unshifted versions of the same key. For example, the upper case "A" key and its unshifted version, the "a" key, may be programmed with different definitions.

Function keys F1-F4, F6-F8, and the PT (Pad Terminator) key have no definitions assigned to them; their default definitions are "undefined." These keys are reserved specifically for programmed definitions. The SEND key (function key F8) is usually programmed with some version of the SEND command.

Section 2

COMMAND STRUCTURE

HOW TO FIND COMMANDS IN THIS MANUAL

The terminal responds to several dozen commands. This manual is organized functionally and each command, with a description of its structure and what it does, is listed in the appropriate section of the manual: the UP and DOWN commands are described in Controlling the Display, the HRULE and VRULE commands in Forms and Form Fillout, and so forth. The first section in which a command appears contains a complete description of the command syntax.

If the presence of certain modes or settings affects the action of the command, these effects are discussed in the relevant section. The TAB command, for example, causes a different action when the terminal is in form fillout mode, and the action of TAB in form fillout mode is discussed in the Forms and Form Fillout section.

In addition to these command descriptions, Appendix F lists all commands and the sections of the manual in which each command is discussed.

THE FORMAT OF COMMANDS

Each terminal command is represented by an English-style ASCII string. In addition to the English-style commands, the graphics commands have counterparts on existing 4010-series terminals and PLOT 10 software. When these commands are sent from the computer, they can be represented using the 4010-style codes. (See the Graphics section for a discussion of the graphics commands and 4010-style codes.)

Terminal commands consist of four parts:

- The command character
- The command keyword
- The command parameters
- The command terminator

The command character is a unique, user-selectable character that does not normally occur in text. This character informs the terminal that the information which immediately follows is a command. The exclamation point, !, is the default command character. The operator or programmer can change the command character by using the COMMAND command. (See Selecting the Command Character later in this section.) We shall use the exclamation point, !, as the command character throughout this manual.

The command keyword is a single word that identifies the command to be executed. This keyword can be spelled out entirely or, if it contains more than three letters, it can be truncated to the first three letters. The keyword must immediately follow the command character; no spaces or other characters are allowed between the command character and the keyword.

The command parameters, if any, follow the keyword. The type and number of parameters depend on the particular command; some commands take no parameters at all. Parameters can be numbers, character strings, or words. A parameter word can be abbreviated to its first letter.

The last character in a command, whether a parameter or the final character of the keyword, is separated from subsequent information by a command terminator. A terminator can be a semicolon, a carriage return, or another command character. If the command is the final string on a line of text, the terminator is a carriage return. If the command is followed by text, a semicolon terminates the command and separates it from the text. If the command is followed by another command, the command character of the following command can serve as the terminator.

Parameters which are characters or character strings must be separated from the keyword and from each other by separators. A separator can be a comma or one or more spaces. The separator between a numeric parameter and the keyword or between a numeric parameter and neighboring alphabetic parameters can be omitted.

Figure 2-1 illustrates the command format.

Consider the following line:

```
!WOR 20 H;THIS IS THE WORKSPACE!MON H!BEL<CR>
```

The ; terminates the !WOR 20 H command. The ! of the !BEL command terminates the !MON H command. The <CR> terminates the !BEL command and the entire line. The string THIS IS THE WORKSPACE, since it is not preceded by a command character, is treated as text and printed in the workspace.

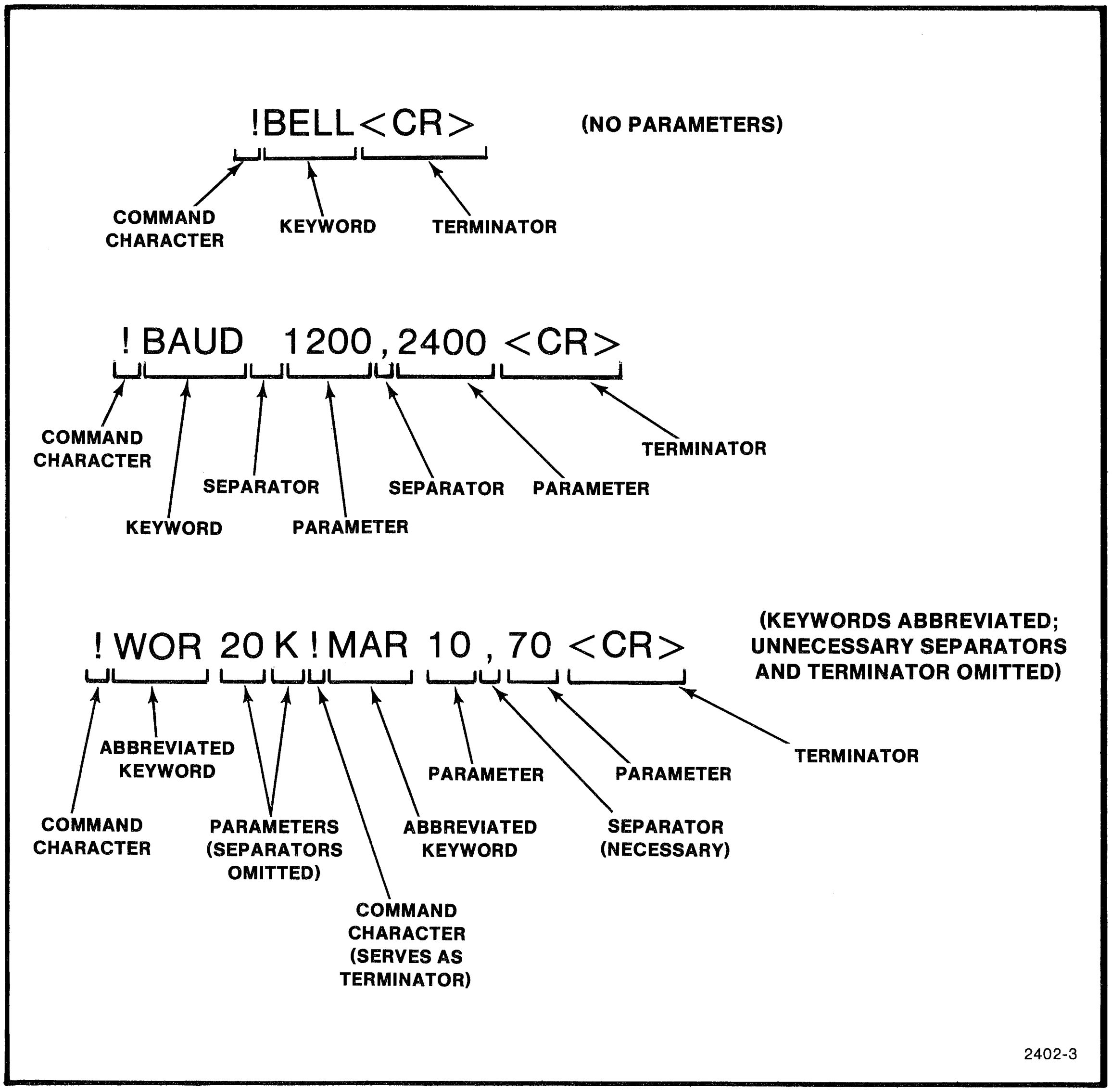


Figure 2-1. Command Format.

Separators followed by + signs can be omitted. The command

```
!RVE + 5,0,– 20,– 110,+ 35,– 110<CR>
```

may be written

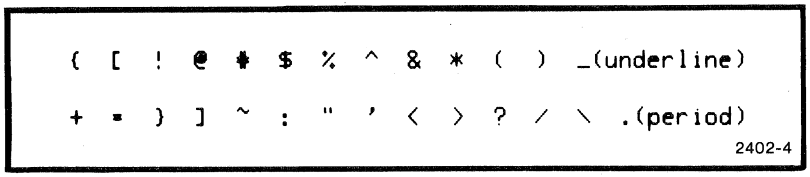
```
!RVE+ 5,0,– 20,– 110,+ 35,– 110<CR>
```

The separator between + 5 and 0 cannot be omitted. The separators followed by — signs cannot be omitted.

Delimited ASCII Strings

Some commands accept delimited ASCII strings as parameters. A delimited ASCII string consists of any string of printing ASCII characters with a delimiter at each end of the string. The delimiters mark the beginning and the end of the delimited string.

The characters which can be used as delimiters are shown in Figure 2-2.



```
{  [  !  @  #  $  %  ^  &  *  (  )  _(underline)

+  =  }  ]  ~  :  "  '  <  >  ?  /  \  .(period)
```

2402-4

Figure 2-2. String Delimiters.

The symbol currently used as the command character cannot be used as a delimiter. The hyphen (-), semicolon (;), and comma (,) cannot be used as delimiters (although their shifted versions can be so used), since these symbols have special uses in the command syntax.

The same symbol must be used for both delimiters of a string. You may write

```
!LEARN F1 [!SEND MOD[ 13<CR>
```

but *not*

```
!LEARN F1 [!SEND MOD] 13<CR>
```

The delimited string must not contain its own delimiter. To set the end-of-line string to the ASCII string * * */*, for example, we could write

```
!EOL * * */*@<CR>.
```

Neither the * nor the / may be used here as a delimiter.

Some commands restrict the length of a delimited string. In general, a delimited string should not contain the command character, except in the LEARN command. See the individual command descriptions for details.

THE SYNTAX OF COMMAND DESCRIPTIONS

Command descriptions which appear in this manual use the following conventions:

- The exclamation point, !, is always used as the command character.

- In a keyword or parameter string which can be abbreviated, the necessary part of the string is written in uppercase; the optional part is written in lower case. For example,

  STOps

  means that any of the strings STO, STOP, or STOPS can be used as the keyword in a STOPS command. Usually the choice will be STO for efficiency or STOPS for readability.

- Expressions in angle brackets, < ... >, are parameter names (except the expression <CR>, which always means carriage return). When a command is given, the parameter name is replaced by one choice from a specified set of valid replacements. The set of valid replacements for the parameter name is listed or described. The DLINE command, for example, is described in this way:

  !DLIne [<count>]<CR>
  where <count> is a positive integer.

- Optional parameters or parameter names are enclosed in square brackets. In the DLINE command noted above,

  [<count>]

  means that the <count> parameter may or may not be specified. Default values are given for all optional parameters.

- Whenever a list appears, with the members of the list separated by vertical bars, |, one element is to be chosen from the list. For example, the FORM command syntax reads:

  !FORm [Yes | No]<CR>

  This means that either Yes or No may be specified, but not both. Neither of these have to be specified. The notation Yes means that Y, YE, and YES are all valid parameter names and define the same command; likewise for No. Thus, !FOR<CR>, !FOR Y<CR>, !FORM YES<CR>, !FOR N<CR>, and !FORM NO<CR> are all valid commands, and there are still others.

- The carriage return, <CR>, is always used as the command terminator when a single command is listed. In particular, in the command descriptions, <CR> always terminates the command.

SELECTING THE COMMAND CHARACTER

When the terminal is shipped from the factory, it recognizes the exclamation point, !, as the command character. The command character can be changed by the computer or the operator by using the COMMAND command. The terminal remembers its command character even when it is RESET or powered off. The only way to change the command character is to give the COMMAND command.

Whenever the terminal receives the command character, it tries to interpret the information immediately following as a command. If this information is not intended to be a command, confusion may result. Therefore, the command character must be selected with care. It should not interfere with normal printing of text or terminal/computer communications.

CAUTION

Symbols such as carriage return, line feed, or space, which are normally used during communications between the terminal and the computer, should NOT be used as command characters.

The command character may vary from one applications program to another. In a text-editing program the exclamation point, !, would be a risky choice for the command character, since this symbol is occasionally used as a punctuation mark. Another symbol, perhaps # or @ , should be chosen.

At the end of a program the command character should always be reset to the exclamation point. In this way, the next user will know the proper command character and be able to command the terminal as needed.

COMMAND

The COMMAND command is used to select a new command character.

Syntax

!COMmand <character><CR>

where <character> is a single ASCII character or a two- or three-digit ASCII Decimal Equivalent (ADE) of an ASCII character.

Action

This command sets the command character to the symbol designated by <character>. If <character> is a single numeral, that character is the new command character. If <character> is a two- or three-digit numeral, that numeral is the ADE of the new command character.

Examples

!COMMAND #<CR> !COM#<CR> !COM 35<CR>	Sets the command character to the number sign, #, whose ADE is 35.
!COM 8<CR>	Sets the command character to the ASCII character 8.
!COM 08<CR>	Sets the command character to the ASCII BS (backspace) character, whose ADE is 08.

Section 3

HOST PROGRAMMING

This section discusses how to use programming language statements to communicate with the terminal. Application programs can be written in any programming language which can display alphanumeric information on the terminal screen and accept data from the terminal.

TEXT AND COMMANDS

All information received by the terminal, whether sent from the computer or typed on the keyboard, can be divided into two categories: commands and text. A command causes the terminal to modify its internal status in some way — perhaps to select a new command character, to redirect text from the computer, etc. Text is information which is printed verbatim on the terminal screen.

The terminal distinguishes between text and commands by the presence of the command character. When the terminal receives the command character, it assumes a command follows and tries to process incoming data as a command. When not processing a command, the terminal treats information as text and displays it in the appropriate text window.

COMPUTER-TO-TERMINAL COMMUNICATIONS

Any programming statement which sends alphanumeric data can be used to send text and commands to the terminal. Common examples are the PRINT statement in BASIC, the WRITE statement in FORTRAN or PASCAL, and the DISPLAY statement in COBOL.

Suppose we are programming in BASIC. The BASIC statement

```
100 PRINT "!WOR 20 K"
```

creates a workspace of 20 lines and directs text from the keyboard into the workspace.

NOTE

When the PRINT statement is executed, the computer sends a <cr> after !WOR 20 K. This <cr> serves as the command terminator.

In contrast, the BASIC statement

```
200 PRINT "WOR 20 K"
```

causes the text WOR 20 K to be displayed in whichever scroll receives text from the computer. The command character in line 100 makes the difference; it indicates to the terminal that the information which follows is a command.

Suppose you wish to initialize the terminal by establishing a 20 line workspace to receive text from the computer, signal the operator by printing the message THIS IS THE WORKSPACE in the workspace, and ring the terminal bell. The BASIC statement

```
100 PRINT "!WOR 20 K;THIS IS THE WORKSPACE!BEL"
```

causes the following events:

- The terminal receives the first !, signaling that a command follows.
- The terminal recognizes the string WOR 20 K; as a valid command and executes it.
- The terminal receives the string THIS IS THE WORKSPACE. As long as the terminal does not see the command character, it treats incoming information as text and prints it in the workspace, which now receives text from the computer.
- The terminal receives the second !, signaling that another command follows.
- The terminal receives the string BEL, followed by the <CR> sent by the computer at the end of the PRINT statement. The terminal recognizes the BEL<CR> as a valid command and executes it.

When the terminal receives information from the computer, it processes that information as it is received. Consider the preceding example:

```
100 PRINT "!WOR 20 K;THIS IS THE WORKSPACE!BEL"
```

The terminal executes the !WOR 20 K; command as soon the ; is received, while continuing to receive information from the computer. The information THIS IS THE WORKSPACE, since it is not a command, is sent to the workspace as soon as the !WOR 20 K; command has been executed. When the terminal receives the <CR> it executes the !BEL<CR> command.

In contrast to this, suppose the following line is typed on the keyboard:

```
!WOR 20 K;THIS IS THE WORKSPACE!BEL<CR>
```

No information is processed until the <CR> is typed. Then the line is processed, displaying the keyboard string in the workspace.

Sending Numeric Parameters

Consider the 4025A VECTOR command:

```
!VEC 100,100 200,100 150,200 100,100<CR>
```

In BASIC, this command can be sent to the terminal in any of the following ways.

1. Include the VECTOR command parameters as alphanumeric data is the PRINT statement

   ```
   495 PRINT "!VEC 100,100 200,100 150,200 100,100"
   ```

   (The PRINT statement provides its own <CR>. This <CR> terminates the VECTOR command.)

2. Send the VECTOR command parameters as data.

   ```
   495 PRINT "!VEC";100,100;200,100;150,200;100,100
   ```

   In this example, as in the preceding one, parameters are separated into coordinate pairs for readability.

3. Define, by host programming, BASIC variables X1= 100, X2= 150, X3= 200, Y1= 100, and Y2= 200. Then use the BASIC statement

   ```
   495 PRINT "!VEC";X1,Y1;X3,Y1;X2,Y2;X1,Y1
   ```

   This method is most versatile, since parameters can be modified by input from the terminal operator or by the program itself.

The 4025A graphic commands are discussed in detail in the Graphics section.

Continuing a Command

Some terminal commands can be continued from one line of code in the host program to the next line of code by inserting a continuation character at the end of the line. There are two cases where this can be done:

- In a VECTOR or RVECTOR command, the ampersand, &, can be inserted after a numeric parameter to continue the command to the next line, provided that the only characters separating the command lines are the ampersand followed by the <CR> or the ampersand followed by <CR/LF>. The BASIC statement

  ```
  100 PRINT “!VECTOR 0,0,175,175,0,175,0,0”
  ```

  can be written as two lines of code:

  ```
  100 PRINT “!VECTOR 0,0&<CR>
  101 PRINT “175,175,0,175,0,0”<CR>
  ```

  for the 4025A, in addition to the ampersand, the parentheses may be used to allow for interline characters that some host systems insert between lines. In this case, the ampersand is followed by left parenthesis, & (, then a seris of characters which may include <CR>, DC1, etx., then right parenthesis,), to resume the command. The BASIC command

  ```
  100 PRINT “!VECTOR 0,0,175,175,0,175,0,0”
  ```

  can be written as two lines of code:

  ```
  100 PRINT “!VECTOR 0,0,&(”. . . interline characters . . .
  101 PRINT “)175,175,0,175,0,0”<CR>
  ```

- In a command which takes a delimited ASCII string as a parameter, the delimited string can be divided into two delimited strings on two consecutive lines of code using the hyphen, -, as a continuation character. The BASIC statement

  ```
  200 PRINT “!LEARN F1 /!SEND ALL!ERA W/13”
  ```

can be written as two lines of code:

```
200 PRINT "!LEARN F1 /!SEND ALL/-"

201 PRINT "/!ERA W/13"
```

The line of text to be continued in this way should NOT be divided between the command character and the keyword, within the keyword, with a numeric parameter, or between a number and its plus or minus sign (if the sign is present).

Individual commands may tolerate minor variations in syntax. See the command descriptions for details.

A Note on Invalid Commands

Since legend has it that not all programs run correctly the first time, some information is in order concerning what to expect from the terminal when it receives data which confuses it.

When the terminal receives an invalid command (that is, a string preceded by the command character but which the terminal cannot recognize as a command), the results depend on the origin of this invalid command. In the following examples the command keyword STOPS is misspelled STEPS:

1. Suppose the invalid command

   ```
   !STEPS 20 40 60<CR>
   ```

   is sent from the computer in the BASIC PRINT statement:

   ```
   100 PRINT "!STEPS 20 40 60"
   ```

   The terminal treats this invalid command as text and prints the entire string, ! STEPS 20 40 60, in whichever scroll receives text from the computer.

2. When the invalid command

```
!STEPS 20 40 60<CR>
```

is typed on the keyboard, an error message is printed and the invalid command is repeated:

```
WHAT?
!STEPS 20 40 60
```

This calls the operator's attention to the source of the error.

3. Suppose this same invalid command is part of a sequence of commands sent from the computer as in the following BASIC statement:

```
100 PRINT "!ERA W!STEPS 20 40 60!BEL"
```

The terminal erases the workspace, prints the text !STEPS 20 40 60 in whichever scroll receives text from the computer, and rings the bell. No error message is given; whatever the terminal cannot recognize as a command is treated as text.

4. If the sequence of commands

```
!ERA W!STEPS 20 40 60!BEL<CR>
```

is typed on the keyboard, all information preceding the invalid command is processed. Then an error message, the invalid command, and the remainder of the line are all printed in the monitor:

```
WHAT?
!STEPS 20 40 60!BEL
```

If the terminal receives a command that requires workspace and no workspace is defined, the command simply ceases to be. Nothing will be executed and no error message will appear.

Displaying a Command File

How does one display a file containing terminal commands so that it can be read, modified, or debugged? There are two ways this can be done:

1. The operator can press the COMMAND LOCKOUT key and then display the file on the screen. When this key is lighted, the terminal treats *all* information, including the command character, as text and prints it in the appropriate scroll.

   Press COMMAND LOCKOUT (light comes on).

   ```
   :
   : (Display file containing ! as the command
   : character, review and edit this file, and
   : return edited file to the computer.)
   :
   ```

   Press COMMAND LOCKOUT again (light goes off).

2. The operator or the computer can change the command character to a symbol which does not appear in the file to be reviewed. In a file which does not contain the symbol #, we might have

   ```
   !COM #<cr> (Change command character to #.)

   !COM #<CR> (Change command character to #.)

    :
    : (Display file containing ! as the command
    : character, review and edit this file, and
    : return edited file to the computer.)
    :

   #COM ! (Reset command character to !.)
   ```

The terminal can also stay execution of commands by using the COPY command (see the Peripherals section).

TERMINAL-TO-COMPUTER COMMUNICATIONS

There are three ways to send information from the terminal to the computer: type into the monitor, give the SEND command, and give the REPORT command.

Typing Into the Monitor

One way to enter information into the computer is to type it into the terminal monitor. If the terminal is in unbuffered mode, information typed into the monitor is sent to the computer character by character, as it is typed. If the terminal is in buffered mode, information typed into the monitor is sent to the computer line by line, as each line is terminated by a carriage return. Buffered and unbuffered modes are discussed in more detail in the System Status and Initialization section.

SEND

A second way to send information to the computer is to first enter that information in the terminal workspace. When the operator or the computer gives the SEND command, all the information in the workspace is sent to the computer.

Syntax

```
!SENd<CR>
```

This command causes all information in the terminal workspace to be sent to the computer.

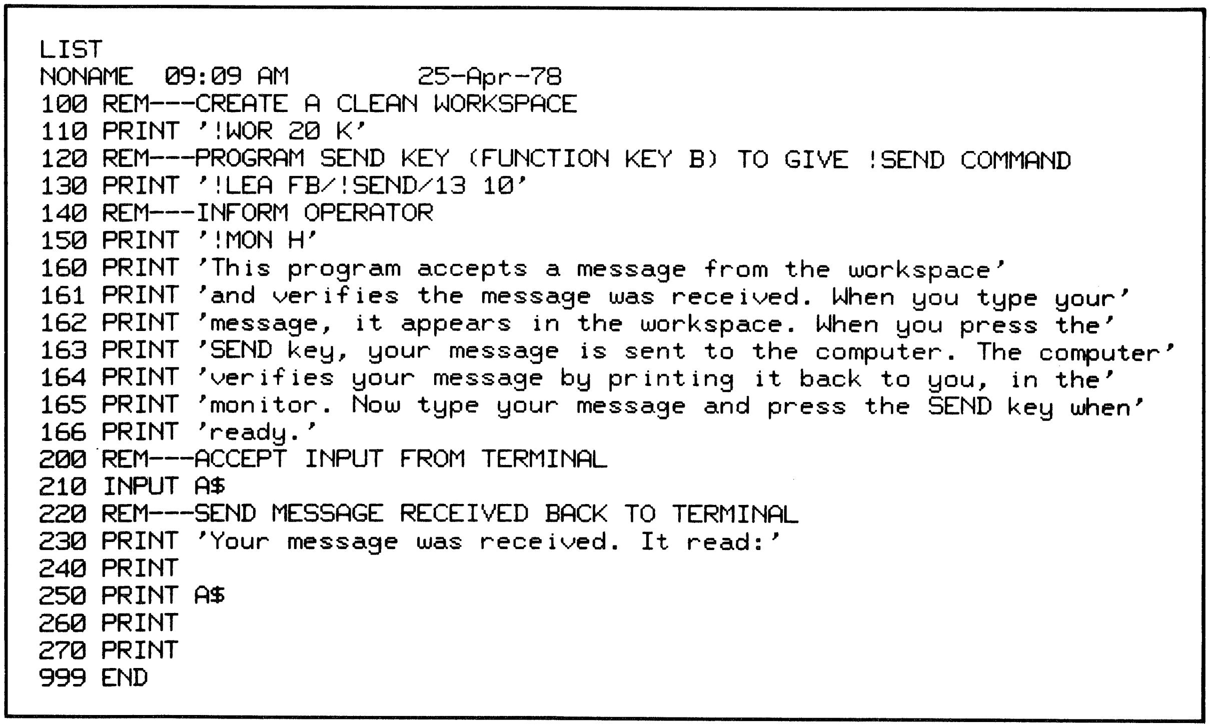
Usually the SEND key is programmed to give the SEND command, so that the operator can send the workspace contents to the computer simply by pressing the SEND key when he is ready.

The SEND command is used in conjunction with whatever input request statement is available in your programming language. In BASIC, for example, the INPUT statement is used; in COBOL, the ACCEPT statement is used.

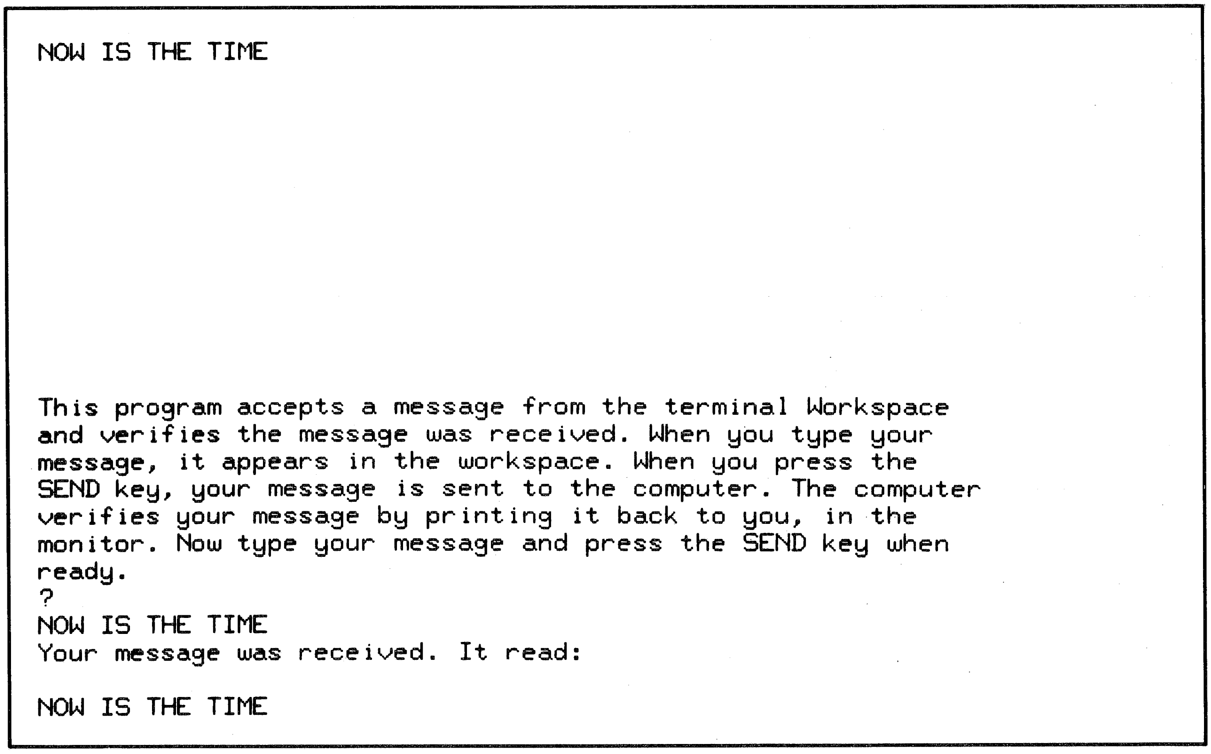
NOTE

The key labelled SEND on the keyboard is not pre-programmed. It may be programmed to give the SEND command using the LEARN command or the LEARN key.

The following program asks the operator to type a one-line message in the workspace and press a key to send this message to the computer. When the computer receives the message, it prints it back in the monitor, so that the operator can verify the message was correctly received.



```
LIST
NONAME  09:09 AM        25-Apr-78
100 REM---CREATE A CLEAN WORKSPACE
110 PRINT '!WOR 20 K'
120 REM---PROGRAM SEND KEY (FUNCTION KEY B) TO GIVE !SEND COMMAND
130 PRINT '!LEA FB/!SEND/13 10'
140 REM---INFORM OPERATOR
150 PRINT '!MON H'
160 PRINT 'This program accepts a message from the workspace'
161 PRINT 'and verifies the message was received. When you type your'
162 PRINT 'message, it appears in the workspace. When you press the'
163 PRINT 'SEND key, your message is sent to the computer. The computer'
164 PRINT 'verifies your message by printing it back to you, in the'
165 PRINT 'monitor. Now type your message and press the SEND key when'
166 PRINT 'ready.'
200 REM---ACCEPT INPUT FROM TERMINAL
210 INPUT A$
220 REM---SEND MESSAGE RECEIVED BACK TO TERMINAL
230 PRINT 'Your message was received. It read:'
240 PRINT
250 PRINT A$
260 PRINT
270 PRINT
999 END
```



```
NOW IS THE TIME











This program accepts a message from the terminal Workspace
and verifies the message was received. When you type your
message, it appears in the workspace. When you press the
SEND key, your message is sent to the computer. The computer
verifies your message by printing it back to you, in the
monitor. Now type your message and press the SEND key when
ready.
?
NOW IS THE TIME
Your message was received. It read:

NOW IS THE TIME
```

NOTE

> *When the SEND command is given from the computer, it* must *be placed in the applications program* before *the input request statement. In BASIC, for example, write*
>
> *100 PRINT “!SEND”*
> *110 INPUT A$*
>
> *Do not write*
>
> *200 INPUT A$*
> *210 PRINT “!SEND’*
>
> *In the latter case the program never executes line 210. It halts at line 200, waiting for data which never comes.*

The use of the SEND command in form fillout applications is discussed in the Forms and Form Fillout section.

REPORT

A third way to send information to the computer is for the computer to issue the REPORT command to the terminal.

Syntax

```
!REPort <device> <CR>
```

where <device> is either the two-digit number 00 or the two-digit number 01.

Action

This command causes the terminal to send a report to the computer. The report has the following format:

```
!ANS <device>,<data field>;
```

The report identifier ANS (for "answer") is followed by one space, the two-digit <device> number, a comma, the <data field>, and finally a semicolon.

The <data field> parameter contains one or more fields, separated from each other by commas. The format of <data field> depends on the value of <device> – that is, on whichever device is reporting. For a given device, however, the format of <data field> is always the same. This allows the applications program to parse <data field> correctly, knowing which device was interrogated.

Examples

1. The command

   ```
   !REP 00<CR>
   ```

   causes the terminal to report the system status block to the computer. This report is in the following format:

   ```
   !ANS 00,<p1>,<p2>;
   ```

   where:

   <p1> is a four-digit decimal number specifying the number of unused blocks of memory. (A block consists of 16 8-bit bytes.)

   <p2> is a three-digit number representing the decimal equivalent of a binary number which specifies the system status byte. The numbers which may be displayed and the condition they represent are:

   004 — monitor present (always true).

   005 — monitor present, buffered mode.

   006 — monitor present, form fillout mode.

   007 — monitor present, form fillout mode, buffered mode.

2. The command

```
!REP 01<CR>
```

causes the terminal to report the workspace cursor status to the computer. This report is in the following format:

```
!ANS 01,<p1>,<p2>,<p3>;
```

where

<p1 is a three-digit decimal number specifying the row of the workspace in which the cursor is located.

<p2> is a three-digit decimal number specifying the column of the workspace in which the cursor is located.

<p3> is a single character, the character displayed at the cursor position. If the cursor is located under an alternate character, such as rulings, the alpha character representing that position is transmitted. If it is a graphics cell, any one of the 128 ASCII characters may be transmitted.

Other values of <device> are possible if the terminal has certain peripheral devices attached. See Peripherals section for details.

The REPORT command can be used for purposes other than straightforward interrogation of the system status block or the workspace cursor.

Some programs may perform considerable processing before sending data to the terminal or requiring input from the terminal. It may be convenient, therefore, to verify that the terminal is alive and listening before proceeding. In BASIC, for example, the applications program can begin with

```
100 PRINT "!REP 00"
110 INPUT A$
```

The program does not proceed until it receives A$. What A$ is, is not important.

As a second example, suppose the applications program is sending large amounts of data to the terminal at relatively high baud rates. It is possible for the computer to overrun the terminal's input buffer, resulting in loss of information. The pair of statements (here, in BASIC)

```
XXX     PRINT "!REP 00"
XXX+ 1  INPUT A$
```

can be inserted occasionally. The program pauses at each input statement and will not continue until it receives input for A$ — that is, until the terminal has processed its entire input buffer and ANSwers the REPort command. What the terminal ANSwers is not important, only that it ANSwers.

The REPORT command is also used to obtain information about peripherals which may be attached to the terminal. Details are contained in the Peripherals section. Appendix E contains a program segment in PASCAL to illustrate how the input from a REPORT command can be processed.

Section 4

PROGRAMMING THE KEYBOARD

The terminal keyboard is programmable; that is, most of the keys can be programmed to generate a character or string of characters other than the default ones. When a key is programmed, the new definition assigned to that key is stored in RAM (Random Access Memory). If the terminal is RESET or powered off, the definition is lost and the key reverts to its default definition.

Key programming enables the operator to give a command or sequence of commands by pressing a single key. During an applications program the operator may log on or log off the computer, change terminal parameters, send information to the computer, page through text, or perform any of several convenient functions just by pressing a key. Key definitions may be part of terminal initialization or may occur at convenient points in a program. A key can have several different definitions in a single program.

All keys can be programmed except the following six keys:

- The rightmost three lighted function keys — TTY LOCK, NUMERIC LOCK/LEARN, and COMMAND LOCKOUT/STATUS. (Neither the shifted nor the unshifted versions of these keys can be programmed.)

- The SHIFT, CTRL, and BREAK keys.

PROGRAMMING A KEY

A key may be programmed with a new definition in one of two ways:

- The operator may use the LEARN key.

- The operator or computer may give the LEARN command.

The LEARN key performs the same action as the LEARN command. The 4025A Operator's Manual describe the use of the LEARN key.

LEARN

Syntax

!LEArn <key> [<string>]<CR>

where <key> designates the key to be programmed and <string> designates the character or character string assigned to the designated key.

Action

This command redefines the key designated by the <key> parameter; whenever this key is pressed, it generates the character string defined by <string>.

Range of Parameters

The <key> parameter may be any of the following:

- A single printing ASCII character.
- A two- or three-digit ADE (ASCII Decimal Equivalent) value from 00 through 127, inclusive. (See the ASCII Code Chart Appendix B.)
- A mnemonic representing a non-ASCII key (function key or cursor/numeric pad key):

F1 — F12	Function keys 1 through 12
S1 — S12	Function keys 1 through 12 with SHIFT depressed
P0 — P9, P., PT	Numeric pad keys and Pad Terminator key

- A "psuedo-ADE value" representing a non-ASCII key:

128	Function Key 1	144	SHIFT-Function Key 1
129	Function Key 2	145	SHIFT-Function Key 2
130	Function Key 3	146	SHIFT-Function Key 3
131	Function Key 4	147	SHIFT-Function Key 4
132	Function Key 5	148	SHIFT-Function Key 5
133	Function Key 6	149	SHIFT-Function Key 6
134	Function Key 7	150	SHIFT-Function Key 7
135	Function Key 8	151	SHIFT-Function Key 8
136	Function Key 9	152	SHIFT-Function Key 9
137	Function Key 10	153	SHIFT-Function Key 10
138	Function Key 11	154	SHIFT-Function Key 11
139	Function Key 12	155	SHIFT-Function Key 12
140	Function Key 13	156	SHIFT-Function Key 13
160	Pad Key 0	172	ERASE
161	Pad Key 1	173	SHIFT-ERASE
162	Pad Key 2	174	BK TAB
163	Pad Key 3		
164	Pad Key 4		
165	Pad Key 5		
166	Pad Key 6		
167	Pad Key 7		
168	Pad Key 8		
169	Pad Key 9		
170	Pad Key .		
171	Pad Terminator Key		

The <string> parameter may be any of the following:

- One or more ADE values.
- One or more pseudo-ADE values.
- One or more delimited ASCII strings.
- Any combination of the above.

If the <string> parameter is omitted, the key is assigned its default meaning (the standard keyboard meaning). The <string> parameter may be any length as long as the terminal's display memory capacity is not exceeded.

Examples

!LEARN # /(End-of-Page)/< CR>
!LEA 35 /(End-of-Page)/< CR>

Redefines the # key (SHIFT-3 key), whose ADE is 35, to generate the parenthetical comment (End-of-Page). The definition of the 3 key is unchanged.

!LEA 35 13< CR>

Redefines the # key to mean carriage return.

!LEA F8 "!SEND MOD;"13< CR>
!LEA 135 "!SEND MOD;"13< CR>

Programs function key F8, whose pseudo-ADE is 135, to give the !SEND MOD command.

!LEA 148 /!WOR!ERA W;READY FOR NEXT PROGRAM/ 7 7 7 /!MON;/13< CR>

Programs the SHIFT-HOME key, whose pseudo-ADE is 148, to direct text from the keyboard into the workspace, erase the workspace, print the message READY FOR NEXT PROGRAM there, ring the terminal bell three times, and return the keyboard to the monitor.

!LEA 148< CR>

Restores the SHIFT-HOME key to its default meaning (undefined).

NOTE

When programming a key to give a command or sequence of commands, **always** *include the ADE 13 as the last character of < string>. This insures that pressing the programmed key causes the command(s) to be executed.*

Special Considerations

When the LEARN command is given from the computer, it may be continued from one line of program code to the next by using a hyphen, -, as a continuation character. This causes the next <CR>, up to one <LF>, and all NULs, RUBOUTs, and SYNCs to be ignored until another character is received. The LEARN command

```
!LEA F3 /THIS COMMAND IS TOO LONG TO FIT ON ONE LINE./ 13 <CR>
```

can be written on two consecutive lines of BASIC program code as follows:

```
100 PRINT "!LEA F3 /THIS COMMAND IS TOO /-"
101 PRINT "/LONG TO FIT ON ONE LINE./ 13"
```

This does not apply to a LEARN command entered from the keyboard. If the command is entered from the keyboard, one simply continues typing until the command is complete. If the command is longer than one line (80 characters), the cursor wraps around to the next line; the command is not terminated until <CR> is pressed.

Since delimited strings may contain only printing ASCII characters, any control characters or non-ASCII characters included in a LEARN command must be encoded using ADEs or pseudo-ADEs outside the delimited string. Thus, the command

```
!LEA $ 13 10<CR>
```

programs the $ key (SHIFT-4 key) to mean <CR><LF>. In contrast, the command

```
!LEA $ /13 10/<CR>
```

programs the $ key to print the ASCII string 13 10.

If one of the ASCII numeral keys (0-9) or the period key (.) is programmed, the corresponding numeric pad key (with the NUMERIC LOCK key lighted) is also programmed. Likewise, if the numeric pad key (with NUMERIC LOCK on) is programmed, the corresponding ASCII numeral or period key is programmed. Programming an ASCII key does not program the corresponding cursor pad key with NUMERIC LOCK off. Likewise, programming the cursor pad key with NUMERIC LOCK off does not program the ASCII key marked with the same symbol.

If the character string assigned to a programmed key includes one or more commands, those commands are executed but not displayed on the screen when the programmed key is pressed.

The <string> parameter may include the CLEAR command, discussed later in this section. Suppose we program the F1 function key as follows:

```
!LEARN F1 /!ERA M!CLEAR!BEL;Goodbye for now.!MON/13<CR>.
```

Pressing F1 causes all of the commands to be executed and the text "Goodbye for now" to be printed in the workspace, even though the CLEAR command is given early in this string.

Function key pseudo-ADE's can be included in the <string> parameter, but those ADE's generate default definitions instead of previously programmed definitions. Consider the command sequence:

```
!LEARN 172 /!ERA W!BEL/13<CR>
!LEARN 128 172<CR>
```

The first LEARN command programs the ERASE key (pseudo-ADE 172) to erase the workspace and ring the bell. The second LEARN comand programs function key F1 to mean the same as the unprogrammed ERASE key.

NOTE

The SEND keys (keys F8 and S8, with pseudo-ADEs 135 and 151, respectively) have no meaning until programmed. Normally, these keys will be programmed to send information to the computer.

Macros and the EXPAND Command

(Requires Option 38)

The EXPAND command is used to execute macros which are defined by the LEARN command.

Syntax

```
!EXPand <macro no.><CR>
```

where <macro> is a macro name (M1, M2, . . ., M16).

Action

This command is used to invoke any macros specified by the LEARN command. EXPAND may be given by the keyboard or the host computer. Thus, a command or series of commands or a string may be sent by the host or the operator by giving the EXPAND command.

Example

```
!EXPand M1<CR>
```

causes the string assigned to the given macro (M1) to be inserted in the input queue in place of the EXPAND command. Macros are umber M1 through M16.

The LEARN Command and the COMMAND Command

Do not confuse programming a key using the LEARN command with selecting a new command character using the COMMAND command. These operations are different.

Programming a key with the LEARN command causes the programmed key to generate a different character or character string than it normally generates. In contrast to this, selecting a new command character does not change the character string generated by any key. Rather, it changes the way the terminal processes the default symbol generated by one particular key. The same key generates the same symbol, but that symbol, when seen by the terminal, now has a different effect.

When the COMMAND command selects a new command character for the terminal, this new selection is stored in the battery-maintained RAM. This means that the terminal remembers the new command character, even when it is turned off or RESET. The only way to change the command character is to give a new COMMAND command. When a key is programmed using the LEARN command, however, the learned definition is lost if the terminal is turned off or RESET, and the key returns to its default definition.

KEY PROGRAMMING AND KEYBOARD LOCKOUT

When a key is programmed, the new definition assigned to that key is generated whenever the key is pressed; however, the default character assigned to that key can still be sent to the terminal. It is not the default character, but the key itself, which generates the new definition.

Suppose we execute the following sequence of commands:

```
!LEA 127 34!LEA 34 /!WOR 20 H K/13<CR>
```

The RUBOUT key (ADE 127) is now programmed to mean quotes, ", and the quotes key (ADE 34) is programmed to mean !WOR 20 H K<CR>. The ASCII quotes character can be sent to the terminal with its usual meaning, either by sending the ASCII quotes character (ADE 34) from the computer or by pressing the RUBOUT key on the keyboard.

It may be desirable to prevent an operator from issuing arbitrary commands to the terminal during an applications program, but still allow him to issue certain specific commands or command sequences. During a form fillout program for example, the operator should not be able to modify the form itself, but should be able to give the SEND MOD command.

Key programming can accomplish this. Suppose ! is the command character. If the computer sends the command

```
!LEARN 33 00<CR>
```

to the terminal, the ! (SHIFT-1) key is programmed to generate the ASCII NUL character. This prevents the operator from using the ! key to generate the command character. Yet the computer can send command characters to the terminal and can program function keys to issue commands when pressed by the operator. Only the operator's ability to issue the command character arbitrarily from the keyboard is impaired. At the proper time, the computer returns control of the terminal to the keyboard by sending the command

```
!LEARN 33<CR>
```

This returns the ! key to its default meaning.

CLEARING KEY DEFINITIONS

To restore a single key to its default definition, use the LEARN command with the <string> parameter omitted. The command

```
!LEARN <key><CR>
```

will restore the <key> key to its default meaning.

CLEAR

To clear all programmed key definitions simultaneously, use the CLEAR command. Either of the commands

```
    !CLEAR<CR>
or
    !CLE<CR>
```

clears all key definitions generated by LEARN commands or by the LEARN key. All keys revert to their default definitions.

Section 5

SYSTEM STATUS AND INITIALIZATION

The terminal has many operating parameters which can be set from the keyboard or from the computer. This allows the terminal to interface with a variety of host systems, as well as run many different applications programs easily and effectively. Some of these parameters (the end-of-line string, for example) must be set when the terminal is first installed and are changed infrequently, if at all. Other parameters (the form fillout mode setting, for example) will be changed more often, perhaps several times within the same program.

Clearly, it is necessary for the host and the applications program to be well informed of the status of these parameters. Since these settings may be changed from the keyboard without the host's knowledge, the first task of any applications program is to initialize the terminal; that is, the terminal must be set to a known and desired state which facilitates execution of the program. When the program is completed, the terminal be returned to a known reference state for the convenience of future users.

Some parameters affect the status of the terminal itself. Other parameters affect the status of communications between the terminal and the host computer. This section first discusses the terminal status commands which determine the status of the terminal itself. These are the COMMAND, WORKSPACE, MONITOR, MARGINS, STOPS, FORM, SNOOPY, and PAD commands. Then the communication status commands, which determine the status of communications between the terminal and the host computer, are discussed. These are the BAUD, PARITY, ECHO, BUFFERED, EOL, REMOTE START STOP, PROMPT, DELAY, FIELD, EOF, DUPLEX, and DISCONNECT commands.

TERMINAL STATUS COMMANDS

COMMAND

The syntax of the COMMAND command is

```
!COMmand <character><CR>
```

where <character> is a single printing ASCII character or the ADE (ASCII Decimal Equivalent) of an ASCII character. The syntax and action of this command were discussed in the Command Structure section; however, some additional comments regarding terminal initialization are in order here.

Since each command to the terminal must be preceded by the command character, the computer must know the command character at all times. Although the terminal operator can discover the command character by pressing the STATUS (SHIFT-COMMAND LOCKOUT) key, the computer cannot do this. Therefore, at the end of each applications program the command character must be set to a reference symbol. This insures the next user proper access to the terminal. The exclamation point, !, is recommended as the reference symbol. It is the command character when the terminal is shipped from the factory. It is also used as the command character throughout this manual and throughout the 4025A Operator's Manuals.

The command character can be changed at the beginning of an applications program, or anytime during the program, by using the COMMAND command. But the program should always reset the command character to the reference character, !, before releasing control of the terminal. Consider a text-editing program. Since the ! symbol is used occasionally as a punctuation mark, one may wish to avoid using it as the command character in this situation. Such a program might begin by choosing another command character, say the @ character, and resetting to ! at the end of the program:

```
!COM @<CR>
   :
   :   (Body of program)
   :
@COM !<CR>
End of execution .
```

WORKSPACE

When the terminal is powered up or RESET, there is no workspace or workspace window, the entire 34-line screen is devoted to the monitor window, and text from both the keyboard and the computer is directed into the monitor. Before an applications program is run, the screen must be initialized:

- Divide the screen into a workspace window and a monitor window to display information from the corresponding scrolls.

- Direct text from the computer and from the keyboard into the appropriate scrolls.

One of the commands used to initialize the screen is the WORKSPACE command.

Syntax

```
!WORkspace [<number>] [ Host ] [ Keyboard ]<CR>
```

where <number> is an integer between 0 and 33, inclusive.

Action

If <number> is included, this command erases the entire display list (the monitor, and if a workspace is defined, the workspace also). The terminal then defines a workspace and allots the top <number> lines of the screen for the workspace window. The remaining 34-<number> lines are used for the monitor window. At least one line is always reserved for the monitor window.

If H (Host) is specified, text from the host computer is directed into the workspace. If K (Keyboard) is specified, text from the keyboard is directed into the workspace. (Commands typed on the keyboard are still displayed in the monitor.)

If only the <number> parameter is specified, text from the keyboard and text from the computer go to the same scrolls as before. A WORKSPACE 0 command directs text from both the keyboard and the computer into the monitor, since this command defines no workspace.

If no parameters are specified, and the command comes from the host computer, a WORKSPACE H command is executed. If no parameters are specified and the command is typed on the keyboard, a WORKSPACE K command is executed.

Examples

!WOR 20 H K<CR>

Erases the display list, reserves the top 20 lines of the screen for the workspace window, and directs text from both the computer and the keyboard into the workspace.

!WOR 25<CR>

Erases the display list, reserves the top 25 lines of the screen for the workspace window. Does not change the destination of text from the computer or of text from the keyboard.

!WOR 0<CR>

Erases the display list and reserves the entire 34-line screen for the monitor window. Directs text from both the computer and the keyboard into the monitor, since no workspace is defined.

!WOR H<CR>

Directs text from the computer into the workspace. Does not erase the workspace or change the position of the workspace cursor.

!WOR<CR>

If this command comes from the computer, it directs text from the computer into the workspace. If the command comes from the keyboard, it directs text from the keyboard into the workspace.

MONITOR

The WORKSPACE command does not allow you to specify which devices (Host, Keyboard) send information to the monitor. The MONITOR command allows you to do this, as well as create text windows.

Syntax

```
!MONitor [<number>] [ Host ] [ Keyboard ]<CR>
```

where <number> is an integer between 1 and 34, inclusive.

Action

If <number> is included, this command erases the entire display list (the monitor, and if a workspace is defined, the workspace also). The terminal then defines a workspace and reserves the top 34-<number> lines of the screen for the workspace window. The remaining <number> lines are used for the monitor window. At least one line is always reserved for the monitor window.

If H (Host) is specified, text from the computer is directed into the monitor. If K (Keyboard) is specified, text from the keyboard is directed into the monitor.

If <number> is the only parameter specified, text from the computer and from the keyboard go into the same scrolls as before. A MONITOR 34 command directs text from both the computer and the keyboard into the monitor, since this command defines no workspace.

If no parameters are specified and the MONITOR command comes from the host computer, a MONITOR H command is executed. If no parameters are specified and the MONITOR command is typed on the keyboard, a MONITOR K command is executed.

Examples

!MON 10 H K<CR>	Erases the display list, creates a monitor window of 10 lines and a workspace window of 24 lines, and directs text from the computer and from the keyboard into the monitor.

!MON 4<CR>

Erases the display list, creates a monitor window of 4 lines and a workspace window of 30 lines. Text from the keyboard and text from the computer go into the same scrolls as before.

!MON 34<CR>

Erases the display list and reserves the entire 34 lines of screen for the monitor window. Directs text from both the computer and the keyboard into the monitor, since no workspace is defined. Equivalent to a WORKSPACE 0 command.

!MON H<CR>

Directs text from the computer into the monitor; does not erase either scroll.

!MON<CR>

If this command comes from the computer, it directs text from the computer into the monitor. If the command comes from the keyboard, it directs text from the keyboard into the monitor.

MARGINS

Workspace margins are set with the MARGINS command. (Monitor margins are always set to columns 1 and 80, and cannot be changed.)

Syntax

```
!MARgins [<left>] [<right>]<CR>
```

where <left> and <right> are integers between 1 and 80, inclusive, and <left> is less than <right>. If only one parameter is specified, it is taken to be the <left> parameter; in this case, the <right> parameter remains unchanged. If both parameters are omitted, <left> and <right> default to 1 and 80, respectively.

Action

This command sets the workspace margins — the left margin to column <left> and the right margin to column <right>.

When the terminal receives a <CR> from the computer or from the keyboard, the cursor moves to column <left>. All cursor movement keys and almost all commands which move the cursor respect the left margin: if the left cursor key is pressed repeatedly, the cursor moves left to column <left>, then wraps around to column 80 of the previous line; the BACKTAB key does not move the cursor past column <left>. (The one exception is the JUMP command. See the Controlling the Display section.)

If a character is typed into the next column beyond column <right>, the terminal bell rings. This is the only action which occurs. If more characters are entered in the workspace, those characters are displayed on the same line, and the cursor continues moving right until either (1) the cursor moves past column 80 and wraps around to the next line, or (2) the terminal receives a <CR> as a signal to begin a new line. In either case, the cursor moves to the left margin in column <left> of the next line.

Examples

!MARGINS 10 70<CR>	Sets the left workspace margin to column 10 and the right margin to column 70.
!MAR 25<CR>	Sets the left margin to column 25; leaves the right margin unchanged.

!MAR<CR>	Sets the left and right margins to their default settings: columns 1 and 80, respectively.

The terminal remembers its right and left margins when it is powered off or RESET.

NOTE

Unless stated otherwise, it is always assumed in this manual that the left margin is set to column 1.

STOPS

Tab stops are set with the STOPS command.

Syntax

```
!STOps [<stop 1>] [<stop 2>] . . . [<stop 16>]<CR>
```

where each <stop n> parameter is a positive integer between 2 and 80, inclusive, and parameters are arranged in increasing order.

Action

This command sets up to 16 tab stops by listing the columns in which stops are defined. Stops are defined in both the workspace and the monitor simultaneously. Only the stops specified are defined; all previous stops are deleted. Stops may be set to the left of the left workspace margin, to the right of the right workspace margin, and between the margins.

If no parameters are specified, all tab stops are cleared.

Examples

!STO 10 20 35 45 60<CR>	Defines monitor and workspace tab stops in columns 10, 20, 35, 45, and 60. No other stops are defined; any previously defined stops are deleted.
!STO<CR>	Clears all tab stops.

The 4025A remembers its tab stops when powered off or RESET.

FORM

The FORM command places the terminal in form fillout mode and removes it from form fillout mode.

Syntax

```
!FORm [ Yes | No ]<CR>
```

Action

The FORM YES command (or equivalent) places the terminal in form fillout mode. The FORM NO command (or equivalent) removes the terminal from form fillout mode. A detailed discussion of form fillout mode is found in the Forms and Form Fillout section.

If no parameter is specified, Y (Yes) is assumed.

Examples

```
!FORM YES<CR>          Places the terminal in form fillout mode.
!FOR Y<CR>
!FOR<CR>

!FORM NO<CR>           Removes the terminal from form fillout mode.
!FOR N<CR>
```

The terminal always powers up and RESETs to FORM NO.

SNOOPY

The terminal has a "snoopy" mode of operation. In snoopy mode, the non-printing ASCII characters (control characters) are represented on the screen by two letter mnemonics. (See Table 5-1.) The RUBOUT (or DELETE) character is represented by a blotch of fine diagonal lines. Entering and leaving snoopy mode is controlled by the the SNOOPY command.

Syntax

```
!SNOopy [ Yes | No ]< CR>
```

If neither parameter is specified, Yes is assumed.

Action

The SNOOPY YES command places the terminal in snoopy mode. The SNOOPY NO command removes the terminal from snoopy mode.

Snoopy mode is useful for troubleshooting and debugging, since it allows the operator to examine *all* ASCII characters received by the terminal, not just printed characters. It is also useful for inserting control characters into text stored in the workspace. Commands are still executed in snoopy mode.

To see the ASCII NUL character printed when examining incoming data, it is necessary to have the terminal parity set to "data." (See the discussion of the PARITY command.)

Examples

```
!SNOOPY YES< CR>          Places the terminal in snoopy mode.
!SNO Y< CR>
!SNO< CR>

!SNOOPY NO< CR>           Removes the terminal from snoopy mode.
!SNO N< CR>
```

The terminal always powers on or RESETs to SNOOPY NO.

Table 5-1

SNOOPY MODE MNEMONICS

Control Character	Snoopy Mode Mnemonic	Control Character	Snoopy Mode Mnemonic
NUL	NU	DLE	DL
SOH	SH	DC1	D1
STX	SX	DC2	D2
ETX	EX	DC3	D3
EOT	ET	DC4	D4
ENQ	EQ	NAK	NK
ACK	AK	SYN	SY
BEL	BL	ETB	EB
BS	BS	CAN	CN
HT	HT	EM	EM
LF	LF	SUB	SB
VT	VT	ESC	EC
FF	FF	FS	FS
CR	CR	GS	GS
SO	SO	RS	RS
SI	SI	US	US
		RUBOUT	▨

PAD

The PAD command is used to place the terminal in keyboard lock mode. This enables the host program to control keyboard operation and data entry.

Syntax

```
!PAD [205/203/209/207]<CR>
```

Action

The PAD 205 command places the terminal in keyboard lock mode. With they keyboard locked, no data or commands may be entered from the keyboard. Any attempt to enter data or commands from the keyboard rings the bell. The PAD 203 command, which can be given only from the computer, removes the terminal from keyboard lock mode. Keyboard lock mode may be exited without the computer PAD 203 command by pressing the BREAK key two times in rapid successsion. The PAD 209 command places the terminal in delete ignore mode to discard delete characters (ADE 127) as they are received. The PAD 207 command places the terminal in full 128-character receive mode.

Examples

!PAD 205<CR>	Places the terminal in keyboard lock mode.
!PAD 203<CR>	Computer only — Removes the terminal from keyboard lock.
!BREAK-BREAK	Keyboard only — Removes the terminal from keyboard lock.
!PAD 209<CR>	Places the terminal in delete ignore mode.
!PAD 207<CR>	Places the terminal in full 128-character receive mode.

COMMUNICATIONS STATUS COMMANDS

BAUD

The simplest communications system consists of a device to transmit information, a device to receive information, and a communications link or "line." The rate at which information is transferred over a communications line is called the "baud rate." This rate is given in bits/second; a baud rate of 1200 means information is transferred at the rate of 1200 bits/second.

During any communication, the rate at which the transmitting device transmits information must not exceed the rate at which the receiving device receives it; otherwise the receiving device will be overrun and information will be lost. If the host computer is sending data to the terminal at 1200 baud, the terminal must be set to receive data at 1200 baud.

The terminal has a "receive baud rate" and a "transmit baud rate." These need not be the same; i.e., the terminal may receive information at a different rate than it transmits information.

The 4025A baud rates are set using the BAUD command.

Syntax

```
!BAUd <transmit> [<receive>]<CR>
```

where both <transmit> and <receive> are chosen from the following list:

```
{0|50|75|110|134|150|300|600|1200|1800|2400|4800|9600}
```

Action

This command sets the transmit baud rate to <transmit> and the receive baud rate to <receive>. A baud rate of 0 means a "times one" external clock is used (requires Option 01).

If <receive> is omitted, it is set equal to <transmit>.

Examples

!BAU 300,1200<CR> Sets the transmit baud rate to 300 baud and the receive baud rate to 1200 baud.

!BAU 2400<CR> Sets both transmit and receive baud rates to 2400 baud.

When the 4025A is turned off or RESET, it remembers the current baud rate.

PARITY

In the ASCII code, each of the 128 ASCII characters is represented by a 7-bit binary number. When a character is transmitted, an eighth bit, called a "parity bit," is also transmitted. Some computers use this extra bit for error checking, some use it as a data bit, and some simply ignore it.

The terminal parity must be set to correspond with that of the computer to which it is connected. This is done by using the PARITY command.

Syntax

```
!PARity [ Even | Odd | None | High | Data ]< CR>
```

If no parameter is specified, the terminal parity defaults to None.

Action

This command sets the terminal parity. If the parity is set to Even, the terminal transmits characters with even parity and checks incoming characters for even parity. If the parity is set to Odd, the terminal transmits characters with odd parity and checks incoming characters for odd parity. If the parity is set to None, the terminal transmits characters with parity bit set to zero; the parity of characters input to the terminal is ignored. If the parity is set to High, the terminal transmits characters with parity bit set to one; the parity of incoming characters is ignored. If the parity is set to Data, the parity bit of each character input to the terminal is treated as data; the parity bit is set to zero on characters ouput from the terminal.

Examples

!PAR E< CR>	Sets the terminal to even parity.
!PAR O< CR>	Sets the terminal to odd parity.
!PAR N< CR>	Sets parity to "none;" the terminal ignores the parity bit on input characters and sets it to zero on output characters.

!PAR H<CR>	Sets parity to "high;" the terminal ignores the parity bit on input characters and sets it to one on output characters.
!PAR D<CR>	Sets parity to "data;" the parity bit is read as a data bit for incoming characters and set to zero on output characters.

The terminal remembers its parity setting when powered off or RESET.

ECHO

When the operator types into the monitor in unbuffered mode, there are two ways that the characters typed may be displayed on the screen: *remote echo* and *local echo*.

In *remote echo* communications, characters typed into the monitor are sent to the computer without being displayed. As the computer receives each character, it "echoes" it back to the terminal. (In some systems, a modem may provide the echo.) It is the received echo, rather than the original transmitted character, that the terminal displays on the screen. In remote echo communications:

- As each character is typed into the monitor, the operator can tell immediately whether the computer has received that character correctly.

- *Selective echo* is possible. The computer can be programmed to decide which characters to echo. In timesharing systems, for example, the computer is usually programmed not to echo a user's password.

In *local echo* communications, as each character is typed into the monitor, the terminal supplies its own echo. It displays each character sent to the computer without waiting for the computer echo. Local echo communications may be used with half duplex communications links, while remote echo requires full duplex communications.

It is important that the terminal be set for the proper echo. If the terminal is set to remote echo and neither the host nor the modem provides an echo, characters typed on the keyboard are not displayed at all. If the terminal is set to local echo and either the host or the modem also provides an echo, characters typed in the keyboard are displayed twice.

The type of echoing which the terminal uses is selected with the ECHO command.

Syntax

```
!ECHo [Local|Remote]<CR>
```

If neither L nor R is specified, L is assumed.

Action

This command selects the echoing used by the terminal when text from the keyboard is directed into the monitor and the terminal is in unbuffered mode.

Examples

!ECH<CR> !ECH L<CR>	Sets the terminal for local echo.
!ECH R<CR>	Sets the terminal for remote echo.

The terminal remembers its ECHO setting, even when power is turned off.

BUFFERED

The terminal can operate either in unbuffered mode or buffered mode. These modes of operation differ in the way that the terminal processes information from the keyboard and from the computer. The terminal powers up in unbuffered mode. It remains in unbuffered mode until placed in buffered mode by the BUFFERED command.

When the terminal is in unbuffered mode, each character typed into the monitor is immediately transmitted to the host. Under these circumstances, it is not possible to locally edit the information displayed in the monitor. As soon as a character appears in the monitor window (if in local echo), it is sent to the computer. Text typed into the workspace is not sent to the computer until the SEND command is given and executed. When the SEND command is executed, all the text in the workspace is sent to the computer in an uninterrupted stream.

When the terminal is in buffered mode, characters entered in the monitor are stored in the keyboard buffer until RETURN is pressed. Anytime before RETURN is pressed, the current line can be edited locally. When RETURN is pressed, the terminal marks the end of the line and stores the line in the transmit buffer. The line remains in the transmit buffer until it is processed. By comparison, each line typed in the workspace is stored there and can be edited locally, even after RETURN is pressed. When the SEND command is given, the entire workspace contents are read into the transmit buffer for processing.

The contents of the transmit buffer are processed line by line on a first-in/first-out basis. To do this, the terminal uses a handshaking process involving prompts (prompt strings) from the computer and EOL (end-of-line) strings from the terminal.

When the computer is ready to receive data, it sends a prompt and delay to the terminal. When the terminal receives this prompt followed by the delay, it knows that the computer has finished its transmission and is ready to receive data. The terminal then processes the oldest (first-in) line in its transmit buffer. Information destined for the computer is sent there and any terminal commands entered from the keyboard are executed. When a line is sent to the computer, an EOL terminates the line. When the computer sees the EOL string, it knows the terminal has finished processing a line and is waiting for another prompt or data from the computer. If the computer has data for the terminal, it sends this out, followed by a prompt; if the computer has no data but wants another line from the terminal, it simply sends a prompt. A more detailed description of each command and its operation follows.

The commands which relate to buffered mode are PROMPT, DELAY, and BUFFERED. The PROMPT command sets the prompt string to be used by the computer to request a line from the terminal. The DELAY command sets the time interval between a computer prompt and a transmitted line, plus sets the time measured after a prompt string to assure that the string is actually a prompt rather than text. These two commands are described on the following pages under PROMPT and DELAY.

The BUFFERED YES and BUFFERED NO commands are used to enter and exit buffered mode and can be invoked either from the computer or keyboard. The effect of the BUFFERED commands and the sequence of buffered mode events differ depending upon the source of the commands, computer or keyboard.

Syntax

!BUFfered [Yes]<CR>	from the keyboard
!BUFfered [Yes]<CR>	from the computer
!BUFfered [Yes];	from the computer

Yes is assumed if not specified.

Action

The BUFFERED YES command puts the terminal in buffered mode regardless of the source of the command or previous unbuffered condition. If previously in unbuffered mode and the !BUF command is given from the keyboard, the output buffer is armed to send the first line placed in the transmit buffer without the need of a host prompt. If already in buffered mode when the keyboard !BUF command is given, there is no change to the original first line condition, and a prompt is required for each additional line in the transmit buffer.

If the computer is the source of the !BUF command, the functions are threefold. First, the !BUF command places the terminal in buffered mode. Second, the computer !BUF command cancels any previous prompt which may have the transmit buffer in an armed condition. This is used prior to communication of any computer commands to prevent the terminal processor from attempting a transmit while in computer command mode. And third, the computer !BUF command places the workspace in keyboard type-ahead. When the terminal is in type-ahead, keyboard characters directed to the workspace are not immediately displayed. Type-ahead prevents interaction between simultaneous workspace display of computer and keyboard information.

When the computer is ready for the terminal to proceed, the prompt string is sent to the terminal. If the defined prompt string is followed by the specified DELAY time (no CR, NUL, SYNC, or other characters), the above condition of type-ahead is cancelled, releasing keyboard data to the workspace, and the transmit buffer is armed for one line.

Syntax

!BUFfered [No]< CR>	from the keyboard
!BUFfered [No]	from the computer
!BUFfered [No];	from the computer

N or No must be specified

Action

The BUFFERED NO command puts the terminal in unbuffered mode and transmits any lines remaining in the transmit buffer. If the keyboard is the source of the !BUF N command, the command is placed in the keyboard buffer. (The keyboard buffer holds keyboard data and is separate from the transmit buffer.) If there are lines or commands in the keyboard buffer awaiting prompts, the keyboard !BUF N command does not execute until the lines or commands are prompted in sequence. If the keyboard buffer is empty, execution of the !BUF N command is immediate. The terminal exits buffered mode and transmits the remaining lines to the host.

If the computer is the source of the !BUF N command, execution is always immediate. The terminal exits buffered mode, transmits any remaining lines to the computer, and executes any commands waiting in the keyboard buffer.

Examples

keyboard !BUF< CR>	Places the terminal in buffered mode. If previously unbuffered, arms transmit buffer.
keyboard !BUF!SEN< CR>	Places the terminal in buffered mode. Sends the workspace to the transmit buffer. If previously unbuffered, transmits one line.
computer !BUF< CR> !BUF;	Places the terminal in buffered mode. Cancels an outstanding prompt. Places the workspace in keyboard type-ahead.
computer < prompt/delay>	Follows the above host command. Waits the delay then arms the transmit buffer. Removes the workspace from type-ahead.

computer !BUF;<prompt/delay>
Places the terminal in buffered mode. Cancels an outstanding prompt. Waits the delay then arms the transmit buffer.

computer !BUF!JUM n n;
Cancels an outstanding prompt. Directs host output to terminal display.

computer <promtp/delay>
Follows the above host command sequence. Waits the delay then arms the transmit buffer. Removes the terminal from type-ahead.

keyboard !BUF N<CR>
Exits buffered if keyboard buffer is empty. Transmits all lines in transmit buffer.

computer !BUF N<CR>
!BUF N;
Exits buffered mode. Transmits all lines in transmit buffer. Executes any buffered keyboard commands.

The computer should always transmit 'display' command sequences (!JUM, !ATT, etc.) to the terminal starting with !BUF and ending with a <prompt/delay>. This minimizes the possibility of keyboard entry interfering with the computer commands. Once the !BUF has executed, there is no chance of interaction.

For example, use: !BUF!JUM!ATT E;ENTER!ATT S!JUM10,20; <prompt/delay>

Computer sequences which include 'output' commands (!SEN, !REP, etc.) should start with !BUF and include a <prompt/delay> prior to the output command. The computer should not use sequences such as !BUF!SEN.

For example, use: !BUF!JUM!ATT E;STOP!ATT S; <prompt/delay> !SEN;

Initialization commands (!DEL, !PRO, !WOR, !MON, !EOL, etc.) should be done prior to displaying computer information and entering buffered mode. A minimum of 500 milliseconds should follow a command to set up the workspace/monitor screen (!WOR n H K) before sending a buffered <prompt> to the terminal. When prompting the terminal, the host should not send in excess of eight prompt strings within one delay time.

Break Function

In addition to the previously described BUFFERED NO commands, the BREAK key can be used to exit buffered mode. When pressed two times in rapid succession, the terminal exits buffered mode, cancels all data in the transmit buffer, and sends a break signal to the computer BRK-BRK should be used only when it is desirable to cancel data in the buffers.

EOL (End-of-Line)

When the terminal sends information to the computer, it sends an end-of-line string at the end of each line of text. This end-of-line string tells the computer where one line of text ends and the next line begins. In buffered mode, it also informs the computer that the terminal has finished current processing tasks and can receive data from the computer. Some computers expect to see <CR> (carriage return) at the end of each line; others may expect to see <CR><LF> (carriage return, line feed) or other strings at the end of each line.

When the operator types text into the monitor destined for the computer, an end-of-line string is inserted whenever RETURN is pressed. When text from the workspace is sent to the computer (with a SEND command), an end-of-line string is inserted at the end of each line of text. (In buffered mode, as the computer requests each line of text from the terminal, the terminal sends that line, and inserts an end-of-line string at the end of the line.) The EOL command is used to set the terminal end-of-line string.

Syntax

```
!EOL [<string>]<CR>
```

where <string> may be:

1. One or more delimited ASCII strings.

2. A sequence of ADE values separated by spaces, or commas.

3. Any combination of 1 and 2.

The end-of-line string defined by this command may not be more than ten characters in length.

If <string> is not specified, it defaults to <CR> (carriage return).

Action

This command sets the end-of-line string which the terminal sends to the computer at the end of each line of text.

Examples

!EOL<CR>
!EOL 13<CR>
Sets the terminal end-of-line string to carriage return, <CR>, with ADE 13.

!EOL 13 10<CR>
Sets the end-of-line string to <CR><LF>.

!EOL /* *$/ 13 10<CR>
Sets the end-of-line string to the ASCII string * * $<CR><LF>.

The terminal remembers its end-of-line string when it is powered off or RESET.

REMOTE START STOP

Under certain circumstances either the host computer or the terminal may be limited as to the number of characters which can be received at a time, especially at high baud rates. When these conditions are known to exist, the terminal can be programmed to stop and restart transmission under host control, and, if the host has the capability, the terminal can send characters to stop or resume host transmission.

Syntax

```
!RSS [Host|Terminal|Neither|Both|Status][ADE STOP|ADE START]<CR>
```

Action

This command enables the terminal or host computer to start and stop host transmission.

Examples

!RSS H<CR> Sets the terminal to respond to host control of data from the terminal to the host with default parameters of DC3 for stop and DC1 for start.

!RSS T<CR> Sets the terminal to control the host, with the default parameters of DC3 for stop and DC1 for start.

!RSS N<CR> Turns off the RSS control of the currently active device.

!RSS B<CR> Sets both terminal and host control with the default parameters of DC3 for stop and DC1 for start.

!RSS S<CR> Checks current status of the RSS control. The terminal will respond with:

```
RSS CONTROL:OFF        (if not active)
RSS CONTROL:BOTH       (if both are active)
RSS CONTROL:HOST       (if host mode is active)
RSS CONTROL:TERMINAL   (if terminal mode is active)
```

PROMPT

In buffered mode, when the host computer is ready to accept another line of text from the terminal, it sends a *prompt* or *prompt string* as a cue for the terminal to transmit another line. Prompt strings vary with the computer and with the program; but the prompt to which the terminal responds must agree with the prompt sent from the computer. The prompt string is set using the PROMPT command.

Syntax

```
!PROmpt [<string>]<CR>
```

where <string> may be:

1. One or more delimited ASCII strings.
2. A sequence of ADE values separated by spaces or commas.
3. Any combination of 1 or 2.

The <string> parameter may not define a string of more than ten ASCII characters.

If <string> is omitted, the prompt string is set to the line feed character, <LF>.

Action

This command sets the prompt string to <string>. In buffered mode, the terminal waits to receive <string> from the computer before processing the next line in its transmit buffer.

Examples

!PRO /* * $/<CR>	Sets the prompt string to * * $. In buffered mode, the terminal must receive this string from the host before it sends a line of text from its transmit buffer.
!PRO 13 10<CR>	Sets the prompt string to <CR><LF>, with ADEs 13 and 10, respectively.

!PRO /* *$/13 10<CR>	Sets the prompt string to * *$<CR><LF>.
!PRO<CR>	Sets the prompt string to the default setting, <LF>.

The terminal remembers its prompt string when RESET or powered off.

DELAY

Sometimes it is desirable that the terminal not respond immediately to a prompt from the computer. If the terminal is executing a SEND command on a rather full workspace and the computer's input buffers are small, it is possible for the transmission to overrun this input buffer. Information is lost and communications are garbled.

The prompt string may be used in other ways as well. Suppose the prompt string is <LF> and the computer is sending a paragraph of straight text to the terminal. There will be many line feeds which are not intended as prompts. If the terminal waits before responding to a <LF>, and another character is received, the terminal knows to cancel the planned response and keep listening to the computer for more text.

The transmission delay is set using the DELAY command.

Syntax

```
!DELay <time><CR>
```

where <time> is a positive integer.

Action

This command sets the transmission delay to <time> milliseconds. In buffered mode, after a prompt is detected, the terminal waits at least <time> milliseconds before transmitting anything back to the computer.

Examples

!DEL 20<CR>	Causes the terminal to wait at least 20 milliseconds before responding to a prompt from the computer.
!DEL 0<CR>	The terminal responds immediately to a prompt from the computer.

The terminal remembers its delay time when it is RESET or powered off.

FIELD

When the terminal, in form fillout mode, sends form fields to the host computer in a SEND operation, the computer must know when a new field begins. This can be arranged in two ways:

1. Fields sent to the computer are preceded by a *field separator* character; each time the computer sees this character it knows a new field immediately follows. If a field has not been completely filled out, only the filled out portion of the field is transmitted; trailing spaces are not sent.

2. Each field is sent in its entirety, including trailing spaces.

The choice of which method to use is determined largely by the programming language used. (See Forms and Form Fillout for details.)

The terminal is instructed how to send form fields to the host by using the FIELD command.

Syntax

```
!FIEld [<character>]<CR>
```

where <character> is a single printing ASCII character, or a 2- or 3-digit ADE between 00 and 127, inclusive.

If no paramter is specified, it is assumed to be NUL.

Action

This command sets the character which precedes fields of a form when they are transmitted to the computer by the terminal. If no value is supplied, then no character is inserted before a field, and trailing spaces are sent. Common choices for the field separator are TAB, CR, and US.

Examples

!FIE@<CR>
!FIE 64<CR>

Sets the field separator to the @ character, with ADE 64. This character precedes each field of a form sent to the computer.

!FIE<CR>

When fields of a form are sent to the computer, no field separator is used. Each field is sent in its entirety, including all trailing spaces.

The terminal remembers the field separator when RESET or powered off.

EOF

(Requires Option 03 or 04)

The terminal can copy a file from one device to another by using the COPY command. (See Peripherals Section.) When this happens, the terminal looks for an end-of-file string to know when to stop the COPY operation.

The end-of-file string is selected using the EOF command.

Syntax

```
!EOF [<string>]<CR>
```

where <string> consists of:

1. One more delimited ASCII strings.
2. A sequence of ADE values separated by spaces or commas.
3. Any combination of 1 and 2.

This command may not define an ASCII string of more than ten characters.

If <string> is not specified, it defaults to /* .

Action

This command sets the end-of-file string. This string marks the end of a file transferred by a COPY command. See the Peripherals section.

Examples

!EOF/$**/<CR>	Sets the end-of-file string to the ASCII string, $**. This string marks the end of a file transferred by a COPY command.

!EOF 27 27 7<CR>	Sets the end-of-file string to <ESC><ESC><BEL>.
!EOF/* */27<CR>	Sets the end-of-file string to * *<ESC>.
!EOF<CR>	Sets the end-of-file string to its default value, /*.

The terminal remembers the EOF setting when RESET or powered off.

DUPLEX

(Requires Option 01)

The terminal with Option 01 may be set for either full duplex or half duplex communications.

Full duplex mode is used with full duplex communication lines, which permit both terminal and host to transmit at the same time. Half duplex is used with half duplex communications lines, over which only one device (terminal or host) can transmit at a time.

Half duplex communications can use either normal or supervisor mode.

In half duplex communications, the termianl can also be set to respond to either "line turnaround only" or "prompt string plus line turnaround" as the prompting condition in buffered mode.

The DUPLEX command is used to set the terminal for half duplex or full duplex communications.

Syntax

```
    !DUPlex [<fulldup> | <halfdup>]<CR>

where  <fulldup> = Full
       <halfdup> = Half [Supervisor | Normal][Line | Prompt]
```

If no parameters are specified, full duplex operation is assumed. If half duplex is chosen but neither Normal nor Supervisor mode is specified, Supervisor mode is assumed. If half duplex is chosen and neither Prompt nor Line is specified, Line is assumed.

Action

This command sets the terminal for either full duplex or half duplex communications. If half duplex is chosen, either Supervisor or Normal mode is chosen. Also, the prompt condition to which the terminal responds in buffered mode is set to either Line (line turnaround only) or Prompt (prompt string plus line turnaround).

Examples

!DUP<CR>
!DUP F<CR>

Sets the terminal for full duplex.

!DUP H<CR>
!DUP H S<CR>
!DUP H S L<CR>

Sets the terminal for half duplex with supervisor. In buffered mode the prompt condition is line turnaround only.

!DUP H S P<CR>

Sets the terminal for half duplex with supervisor. In buffered mode the prompt condition is the prompt string plus line turnaround.

!DUP H N<CR>
!DUP H N L<CR>

Sets the terminal for half duplex normal. In buffered mode the prompt condition is line turnaround only.

!DUP H N P<CR>

Sets the terminal for half duplex normal. In buffered mode the prompt condition is the prompt string plus line turnaround.

The terminal remembers its duplex setting when RESET or powered off.

DISCONNECT

(Requires Option 01)

Syntax

```
!DISconnect<CR>
```

Action

This command sends a signal to the modem, causing it to disconnect the terminal from the communications line. (The terminal turns off the "data terminal ready" signal on the RS-232 interface for about one second. This causes the modem to disconnect from the communications line.)

Example

!DIS<CR>	Disconnects the terminal from the communications line.

BREAK FUNCTIONS

The BREAK key is used to signal an interrupt to the computer and to terminate a variety of local operations regarding buffered mode and peripheral functions. The effects of a single press of the BREAK key differ from two presses of the BREAK key as follows:

BREAK BREAK. The RS232 TDATA communication line is held active for 350 milliseconds, buffered mode is exited, transmit and receive buffers are cancelled, keyboard lock is exited, and a COPY operation is terminated. In addition, the terminal terminates a DIRECTORY operation and discontinues a multiple HCOPY command.

BREAK. The RS232 TDATA communication line is held active for 350 milliseconds. Internal terminal operations are not affected.

STATUS MESSAGES

In addition to the commands which set the terminal parameters and communications parameters, there are three "status" messages which display, on the screen, information about the parameter settings and internal status of the terminal. These are the STATUS message, the SYSTAT message, and the system RAM TEST message. If the terminal has a graphics memory (Option 25 or 26) a GTEST (Graphic Test) message is also available.

The STATUS Key and the STATUS Message

At any time, the operator may press the STATUS (SHIFT-COMMAND LOCKOUT) key to get a brief STATUS message. This message is displayed in the monitor, without disturbing the contents of the workspace. The STATUS message shows whether the terminal is in buffered or unbuffered mode, the command character, and the number of unused blocks of terminal memory. (A block consists of 16 eight-bit bytes. One block holds at most 14 characters.) Two of these status messages are shown in Figure 5-1.

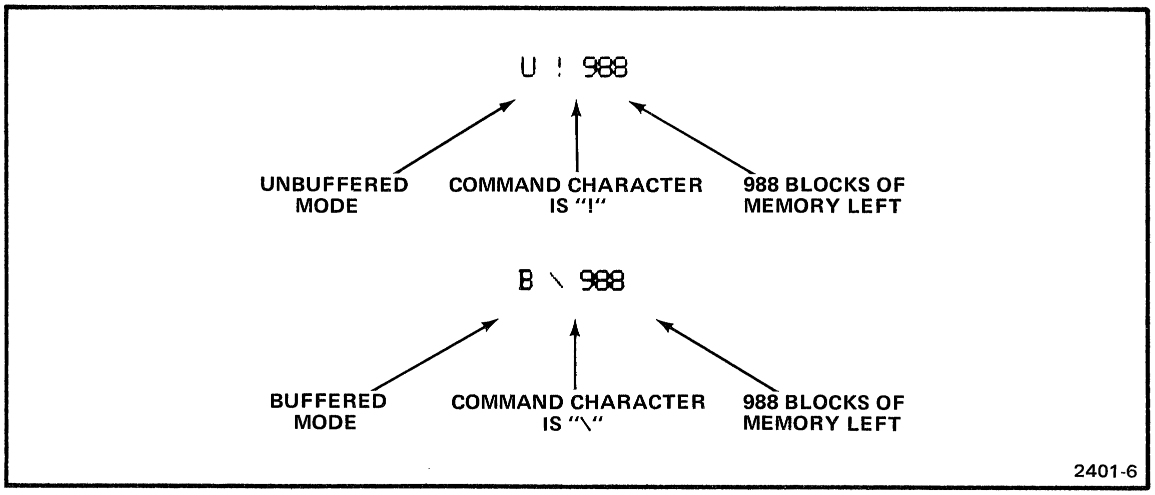


Figure 5-1. STATUS Messages

SYSTAT and the SYSTAT Message

The SYSTEM STATUS, or SYSTAT message lists most of the parameter settings discussed in this section. The SYSTAT command displays the SYSTAT message on the terminal monitor window.

Syntax

```
!SYStat<CR>
```

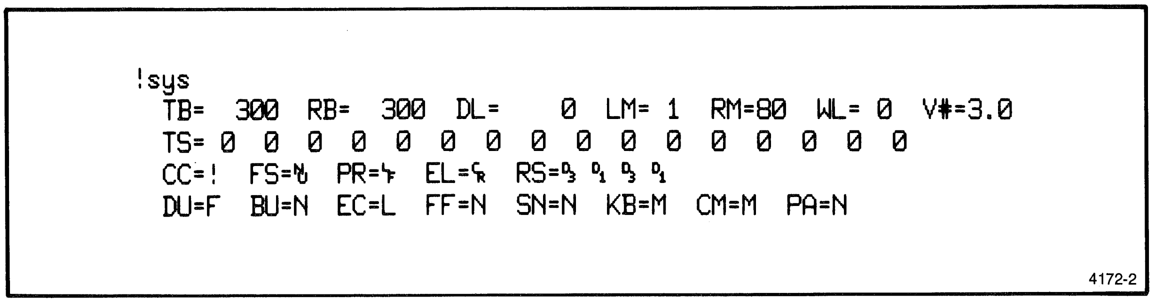
SYSTAT Parameters

The SYSTAT message lists the following parameters, using the abbreviations shown.

TB — Transmit baud rate
RB — Receive baud rate
DL — Delay time
LM — Left margin
RM — Right margin
WL — Number of workspace lines displayed on the screen
V# — Firmware version number
TS — Tab stops
CC — Command character
FS — Field separator
PR — Prompt string
EL — End-of-line string
RS — Remote Start Stop characters
DU — Duplex (DU= F means full duplex, DU= H means half duplex)
BU — Buffered mode (Y means buffered, N means unbuffered)
EC — Echo (EC= R means remote echo, EC= L means local echo)
FF — Form fillout mode (Y means yes, N means no)
SN — Snoopy mode (Y means yes, N means no)
KB — Keyboard (KB= M means text typed on the keyboard is directed to the monitor, KB= W means text from the keyboard is sent to the workspace)
CM — Communications line (CM= M means text from the communications line is directed to the monitor, CM= W means such text is sent to the workspace)
PA — Parity (N means none, D means data, E means even, O means odd, H means high)

If the terminal contains Option 01 (Half Duplex) and is set for half duplex communications, the DU field may contain one or two additional letters. See the DUPLEX command description earlier in this section for details.

If a parameter is set to an ASCII control character, the two-letter mnemonic for that character is shown in the parameter setting. The SYSTAT message is illustrated in Figure 5-2.



```
!sys
 TB=  300  RB=  300  DL=    0  LM= 1  RM=80  WL= 0  V#=3.0
 TS= 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 CC=!  FS=NU  PR=LF  EL=CR  RS=D3 D1 D3 D1
 DU=F  BU=N  EC=L  FF=N  SN=N  KB=M  CM=M  PA=N
```

4172-2

Figure 5-2. SYSTAT Message.

When the terminal is turned off or RESET, it remembers some of the parameter settings in the SYSTAT message, and resets others to default settings. Those settings which are remembered are: TB, RB, DL, LM, RM, TS, CC, FS, PR, EL, EF, DU, EC, and PA.

When the terminal is powered up or RESET:

- WL = O (There is no workspace defined.)
- BU = N (The terminal is in unbuffered mode.)
- FF = N (The terminal is not in form fillout mode.)
- SN = N (The terminal is not in snoopy mode.)
- KB = M and CM = M (Both the keyboard and the computer direct text to the monitor.)

The V# setting will not change unless a different firmware version is installed in the terminal.

TEST

The command:

```
!TEST<CR>
```

or

```
!TES<CR>
```

causes the terminal to run a program which checks whether the terminal memory and display are operating properly. The following actions occur:

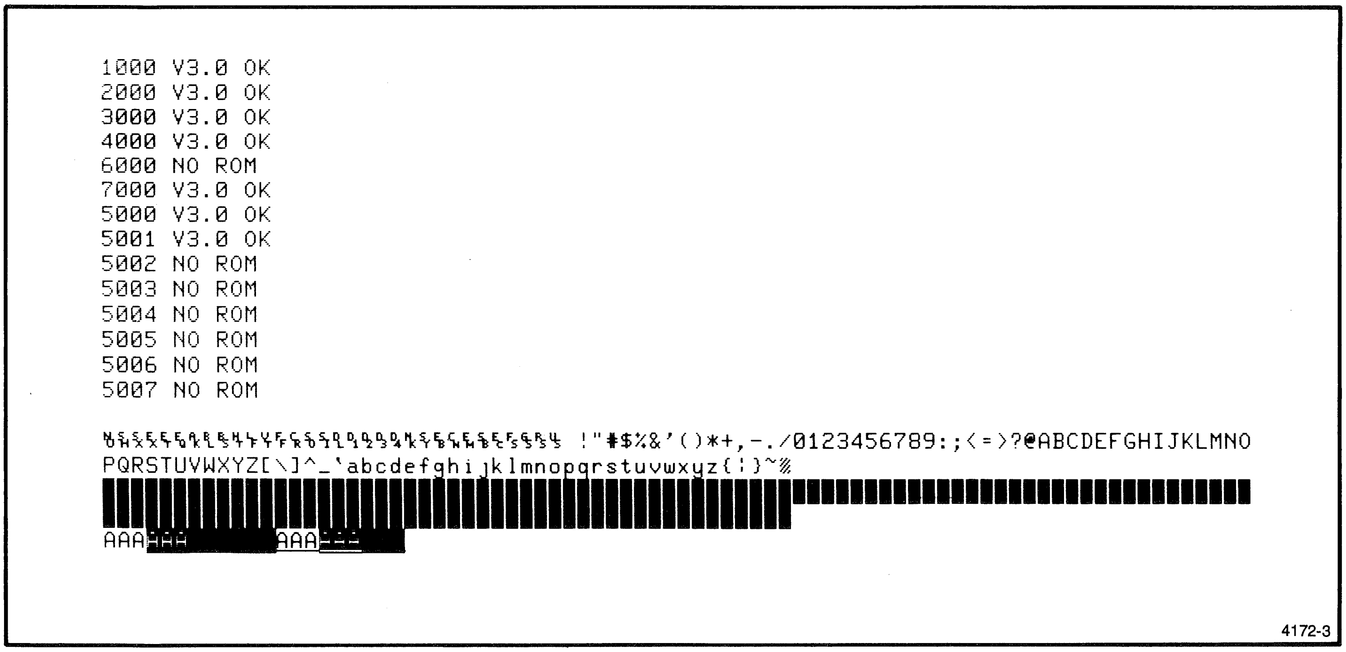
- The terminal erases the entire display list and creates a 34-line monitor window.
- System ROM (Read Only Memory), system RAM (Random Access Memory), and display RAM are checked. Each possible ROM location is displayed with version number. "OK" is displayed if the checksum is correct, or "BAD", along with correct checksum, if incorrect, or "NO ROM" if non is installed.
- After the memory test, the lights on the four lighted function keys are turned on, all 128 ASCII characters are displayed in the monitor in snoopy mode, and all Font 1 characters (ruling characters) are displayed. (If this character set is not installed, each of its characters is displayed as a dot matrix with every dot turned on.)
- After the two character sets are displayed, the visual attributes are diplayed.
- At the end of the test, the lights on the function keys are turned off and the bell is rung.

Should the test reveal a failure in the system RAM, the message "RAM ERROR" appears. In that case, Tektronix service personnel should be called.

NOTE

Running this test destroys any text or key definitions which may have been stored in the terminal memory.

An example of the display created by a successful TEST is shown Figure 5-3.



```
1000 V3.0 OK
2000 V3.0 OK
3000 V3.0 OK
4000 V3.0 OK
6000 NO ROM
7000 V3.0 OK
5000 V3.0 OK
5001 V3.0 OK
5002 NO ROM
5003 NO ROM
5004 NO ROM
5005 NO ROM
5006 NO ROM
5007 NO ROM

␀␁␂␃␄␅␆␇␈␉␊␋␌␍␎␏␐␑␒␓␔␕␖␗␘␙␚␛␜␝␞␟ !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNO
PQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{¦}~▒

AAAAAA      AAA---
```

4172-3

Figure 5-3. !TEST<CR> Results.

GTEST

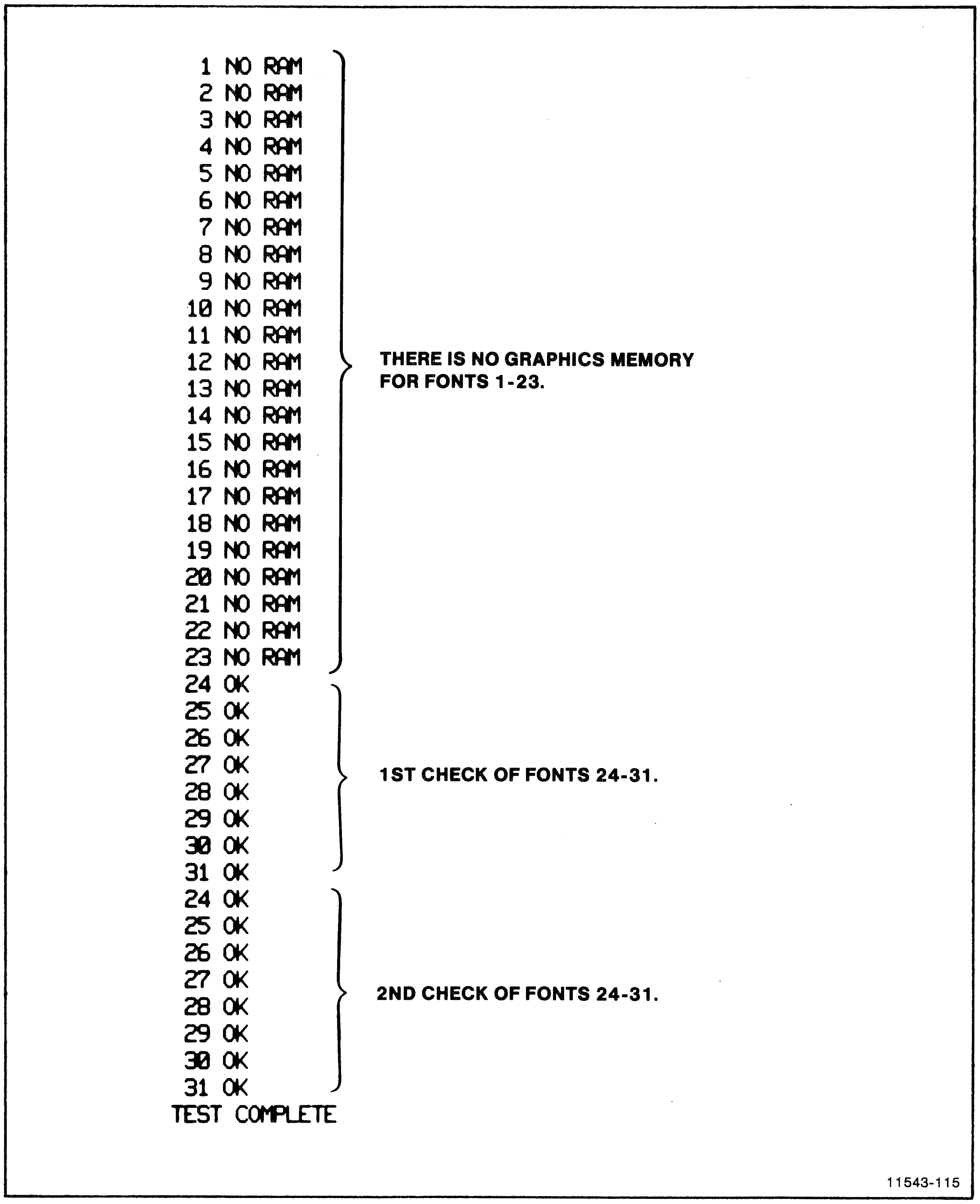
(Requires Option 25 or 26)

If the terminal has a Graphics Memory Option (Option 25 or 26), the command:

```
      !GTEST<CR>
or
      !GTE<CR>
```

when executed, erases the entire display list and a 34-line (screen) monitor window is created. The terminal then tests the graphic memory. After a delay of about 15 seconds while it performs the test, the monitor displays the test results, starting with RAM 1 and proceeding to RAM 31. If no RAM is installed in a particular board location, a "NO RAM" message is displayed. If RAM is installed, each character is tested twice (each bit is tested for both 1 and 0). If the RAM passes the tests "OK" is displayed for each of these two tests. If the RAM for a particular character set fails the test, the "BAD RAM" message is displayed, along with an error code for use by Tektronix service personnel.

A sample display of a successful GTEST is shown in Figure 5-4.



```
 1 NO RAM
 2 NO RAM
 3 NO RAM
 4 NO RAM
 5 NO RAM
 6 NO RAM
 7 NO RAM
 8 NO RAM
 9 NO RAM
10 NO RAM
11 NO RAM
12 NO RAM
13 NO RAM
14 NO RAM
15 NO RAM
16 NO RAM
17 NO RAM
18 NO RAM
19 NO RAM
20 NO RAM
21 NO RAM
22 NO RAM
23 NO RAM
24 OK
25 OK
26 OK
27 OK
28 OK
29 OK
30 OK
31 OK
24 OK
25 OK
26 OK
27 OK
28 OK
29 OK
30 OK
31 OK
TEST COMPLETE
```

THERE IS NO GRAPHICS MEMORY FOR FONTS 1-23.

1ST CHECK OF FONTS 24-31.

2ND CHECK OF FONTS 24-31.

Figure 5-4. !GTEST<CR> Results.

Section 6

CONTROLLING THE DISPLAY

Before information is displayed on the terminal screen, decisions must be made regarding the set-up of the screen: how the screen's 34-line display is to be divided between the workspace window and the monitor window; which scroll is to receive text from the computer and which from the keyboard; and margins and tab stops. The commands which set these parameters are discussed in the System Status and Initialization section. We assume here that these parameters have been set. Throughout this section we assume the left workspace margin is set to column one.

THE CURSOR COMMANDS

There are two cursors — the workspace cursor and the monitor cursor. Only one of these is visible at a given time. Since, in either window, the cursor indicates the position at which new information will be printed on the screen, one may wish to change the cursor position at various times.

The programmer uses commands to position the cursor at a desired location. (The operator may give these same commands from the keyboard, or use the corresponding keys.) The commands which affect the cursor position are the cursor commands (JUMP, UP, DOWN, RIGHT, LEFT) and the tab commands (TAB, BACKTAB). In addition, even though there is no "HOME" command corresponding to the HOME key, the JUMP command can be used to simulate the action of the HOME key. (See discussion of the JUMP command.)

NOTE

If a cursor movement command, tab command, or scrolling command is typed on the keyboard and text from the keyboard is directed into the monitor, execution of the command inserts a line just below the line on which the command is typed.

JUMP

(Workspace only)

Syntax

```
!JUMp [<row> [<column>]]<CR>
```

where <row> is a positive integer, and <column> is a positive integer not greater than 80. If only one parameter is specified, it is assumed to be the <row> parameter. If neither parameter is specified, both <row> and <column> default to one.

Action

This command positions the workspace cursor in the row and column of the workspace designated by <row> and <column>, respectively.

Picture the workspace scroll as a long table with an indeterminate number of rows, each row having 80 columns (Figure 6-1). The topmost row in the workspace, (whether it contains text or is blank) is labeled row 1, the next row is row 2, and so forth. In each row, columns are labeled column 1, column 2, . . ., column 80. This establishes an *absolute coordinate system* in the workspace scroll. Portions of this scroll may be visible in the workspace window.

The JUMP command moves the workspace cursor to the specified row and column of the workspace, expressed in these absolute workspace coordinates. The destination of the cursor does not depend on its current location. (This is in contrast to the other cursor movement commands, whose parameters specify positions relative to the current cursor position.)

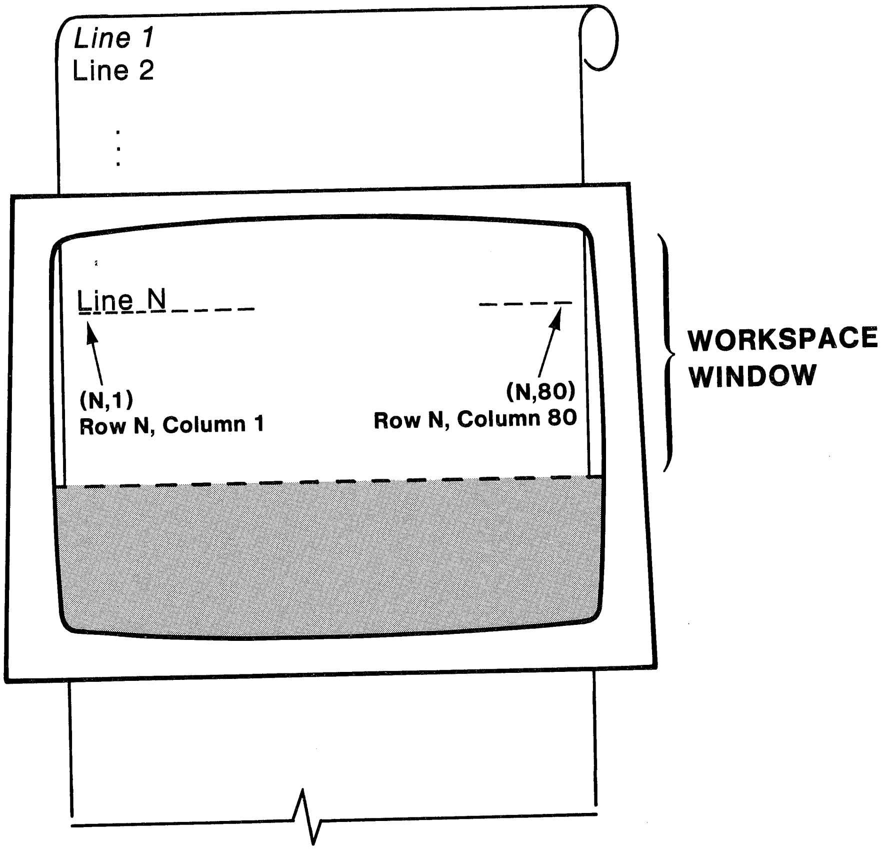
If the JUMP command moves the cursor to a line not visible in the workspace window, the workspace rolls up or down to display the line to which the cursor moves.

If the <row> parameter specifies a row of the workspace below the bottom of the workspace window, the workspace rolls up and stops with the line containing the cursor at the bottom of the window. If <row> exceeds the current number of lines in the workspace, blank lines are created at the bottom of the workspace and the <row>-th row is displayed as the last row in the workspace window.

If the < row> parameter specifies a row of the workspace above the top of the workspace window, the workspace rolls down, stopping with the row containing the cursor at the top of the window.

NOTE

This command applies only, and always, to the workspace cursor. It is not necessary for the workspace to receive text from the computer or the keyboard for this command to move the workspace cursor. When the workspace cursor next appears, it appears at the location specified in the JUMP command (assuming no other instructions which affect the workspace cursor location have been given to the terminal meanwhile).



2402-9

Figure 6-1. The Workspace Window and the Workspace Scroll.

Examples

1. The command

```
!JUM 3,10<CR>
```

moves the workspace cursor to row 3, column 10.

2. Either of the commands

```
!JUM 3<CR>
!JUM 3,1<CR>
```

moves the workspace cursor to row 3, column 1.

3. Any one of the commands

```
!JUM<CR>
!JUM 1<CR>
!JUM 1,1<CR>
```

moves the workspace cursor to row 1, column 1. Each of these commands is equivalent to pressing the HOME key when the workspace cursor is visible and the terminal is not in form fillout mode.

UP

Syntax

```
!UP [<count>]<CR>
```

where <count> is a positive integer. If <count> is not specified, it defaults to one.

Action

This command is equivalent to pressing the up cursor key (pad key 8, marked ↑) <count> times.

This command can be used to move either the workspace cursor or the monitor cursor. If the command is typed on the keyboard, it moves the cursor in that scroll which receives text from the keyboard. If the command is sent from the computer, it moves the cursor in that scroll which receives text from the computer.

Suppose text from the computer is printed in the workspace and the command

```
!UP <count><CR>
```

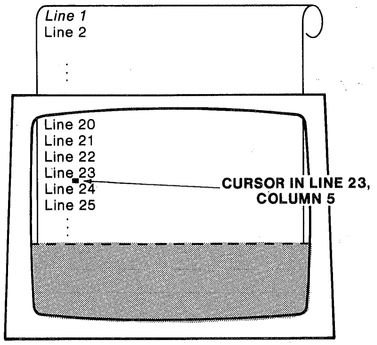
is sent from the computer. This command moves the workspace cursor up <count> lines from its current position, leaving the column location unchanged.

If <count> is large enough to move the cursor to a line not visible in the workspace window, the workspace rolls down so that the line which the cursor moves to is the top line in the window. However, the cursor will not move past the first line of the workspace, regardless of how large <count> is.

If text from the computer is printed in the monitor and this command is sent from the computer, it has the same effect on the monitor cursor.

Examples

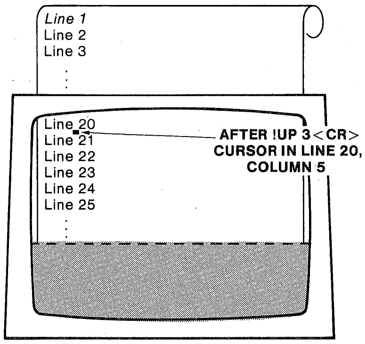
Suppose text from the computer is printed in the workspace, with the cursor in line 23, column 5:



1. The command

   !UP 3<CR>

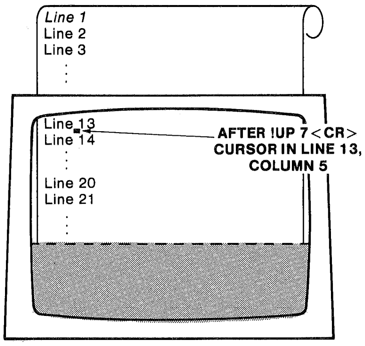
   positions the cursor in line 20, column 5.



2. The subsequent command

   !UP 7<CR>

   causes the workspace to roll down and positions the cursor in line 13, column 5.



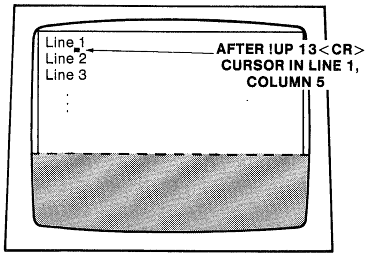
3. The subsequent command

```
!UP 13<CR>
```

rolls the workspace down, leaving the cursor in column 5 of line 1. Since the workspace will not scroll past the first line, the commands

```
!UP 14<CR>
!UP 15<CR>
.
.
.
```

each have the same effect.



DOWN

Syntax

```
!DOWn [<count>]<CR>
```

where <count> is a positive integer. If <count> is not specified, it defaults to one.

Action

This command is equivalent to pressing the down cursor key (pad key 2, marked ↓) <count> times.

This command can be used to move either the workspace cursor or the monitor cursor. If the command is typed on the keyboard, it moves the cursor in that scroll which receives text from the keyboard. If the command is sent from the computer, it moves the cursor in that scroll which receives text from the computer.

Suppose text from the computer is printed in the workspace and the command

```
!DOW <count><CR>
```

is sent from the computer. This command moves the workspace cursor down <count> lines from its current position, leaving the column location unchanged.

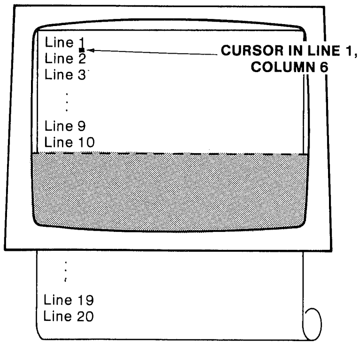
If <count> is large enough to move the cursor to a line not visible in the workspace window, the workspace rolls up until the line which the cursor moves to is at the bottom of the window. If <count> is large enough to move the cursor past the last line in the workspace, enough blank lines are created at the bottom of the workspace to accommodate this command.

If text from the computer is printed in the monitor and this command is sent from the computer, it has the same effect on the monitor cursor.

Pressing the LINE FEED key <count> times has the same effect on the cursor. Pressing this key also generates the ASCII Line Feed character, while pressing the down cursor key does not.

Examples

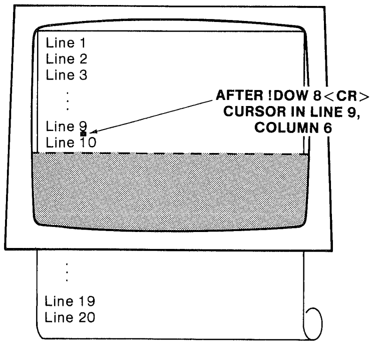
Suppose a workspace window of ten lines is defined, and the workspace contains 20 lines of text (some of which may be blank). Suppose also that line 1 is the top line in the workspace window and the cursor is in line 1, column 6.



1. The command

   !DOW 8<CR>

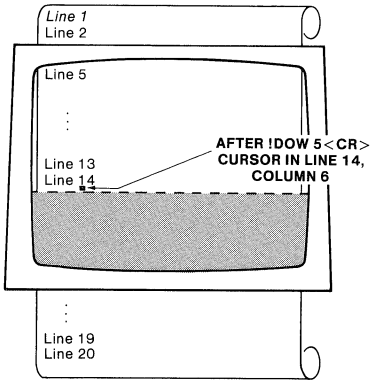
   moves the cursor down eight lines to line 9, column 6. No roll up occurs.



2. The subsequent command

   !DOW 5<CR>

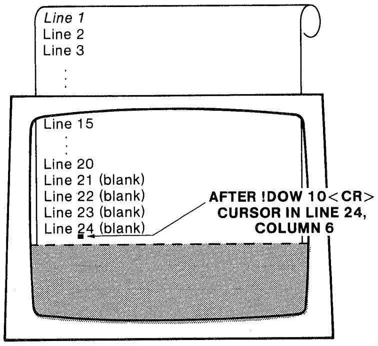
   moves the cursor to line 14, column 6; the workspace rolls up four lines.



3. The subsequent command

   !DOW 10<CR>

   adds four blank lines at the bottom of the workspace and rolls the workspace up 10 lines. The cursor stops in the last blank line created, at the bottom of the workspace window.



RIGHT

Syntax

```
!RIGht [<count>]<CR>
```

where <count> is a positive integer. If <count> is not specified, it defaults to one.

Action

This command is equivalent to pressing the right cursor key (pad key 6, marked →) <count> times.

This command can be used to move either the workspace cursor or the monitor cursor. If the command is typed on the keyboard, it moves the cursor in that scroll which receives text from the keyboard. If the command is sent from the computer, it moves the cursor in that scroll which receives text from the computer.

Suppose text from the computer is printed in the workspace and the command

```
!RIG <count><CR>
```

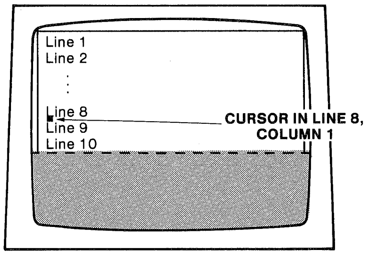
is sent from the computer. This command moves the workspace cursor <count> columns to the right.

If <count> is large enough to move the cursor beyond column 80, the cursor wraps around to the left margin of the next line and continues moving right a total of <count> columns. If this action requires the cursor to move to a line which is not visible in the workspace window, the workspace rolls up so that the line in which the cursor stops is the bottom line in the window. If this command requires the cursor to move beyond the last line of the workspace, enough blank lines are created at the bottom of the scroll to accommodate this command.

If text from the computer is printed in the monitor and this command is sent from the computer, it has the same effect on the monitor cursor.

Example

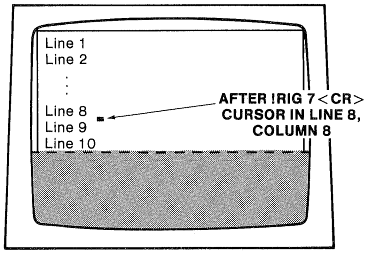
Suppose there is a workspace window of ten lines, with ten lines of text in this window. The left margin is set at column 1 and the cursor is in column 1 of line 8.



1. The command

   !RIG 7<CR>

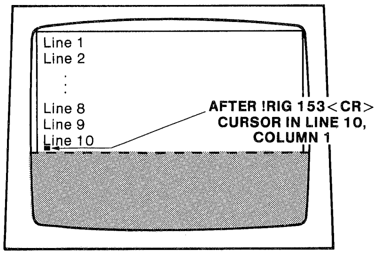
   moves the cursor right seven columns to column 8 of line 8.



2. The subsequent command

   !RIG 153<CR>

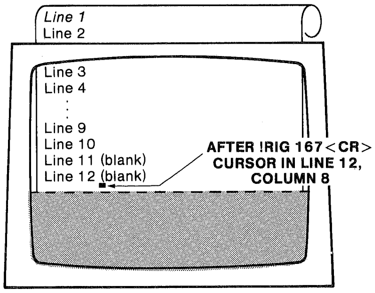
   moves the cursor through the remaining 73 columns of line 8 to column 1 of line 9, then through the 80 columns of line 9 to column 1 of line 10. No roll up occurs.



3. The subsequent command

   !RIG 167<CR>

   moves the cursor through the 80 columns of line 10, creates a blank line 11 and moves the cursor through the 80 columns of line 11, creates a blank line 12 and moves the cursor through seven columns to column 8 of line 12. The workspace rolls up to display line 12 as the last line in the workspace window.



LEFT

Syntax

```
!LEFt [<count>]<CR>
```

where <count> is a positive integer. If <count> is not specified, it defaults to one.

Action

This command is equivalent to pressing the left cursor key (pad key 4, marked ←) <count> times.

This command can be used to move either the workspace cursor or the monitor cursor. If the command is typed on the keyboard, it moves the cursor in that scroll which receives text from the keyboard. If the command is sent from the computer, it moves the cursor in that scroll which receives text from the computer.

Suppose text from the computer is printed in the workspace and the command

```
!LEF <count><CR>
```

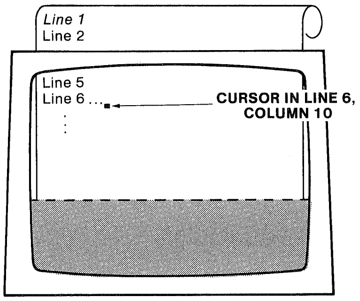
is sent from the computer. This command moves the workspace cursor <count> columns to the left.

If <count> is large enough to move the cursor to the left of the left margin, the cursor wraps around to column 80 of the preceding line and continues moving left a total of <count> columns. If this action requires the cursor to move to a line which is not visible in the workspace window, the workspace rolls down so that the cursor stops in the top line of the window. However, the cursor will not move above the first line in the workspace. Thus this command does not insert blank lines at the top of the workspace.

If text from the computer is printed in the monitor and this command is sent from the computer, it has the same effect on the monitor cursor.

Examples

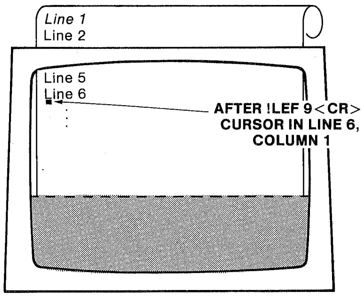
Suppose a workspace is defined and the cursor is visible in column 10 of line 6.



1. The command

   !LEF 9<CR>

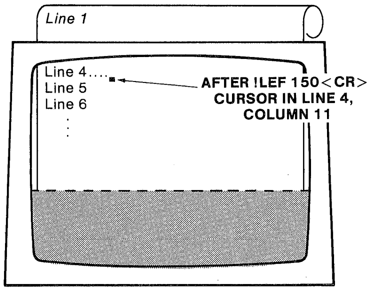
   moves the cursor to column 1 of line 6.



2. The subsequent command

   !LEF 150<CR>

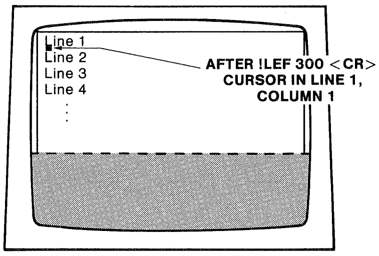
   moves the cursor through the 80 columns in line 5, rolls down the workspace to display line 4, and moves the cursor through the rightmost 69 columns in line 4. The cursor stops in column 11 of line 4.



3. The subsequent command

   !LEF 300<CR>

   moves the cursor through the leftmost ten columns in line 4, then through the 80 columns in each of lines 3, 2, and 1, rolling the workspace down to display these lines. The cursor stops at column 1 of line 1.



THE TAB COMMANDS

TAB

Syntax

```
!TAB [<count>]<CR>
```

where <count> is a positive integer. If <count> is not specified, it defaults to one.

Action

This command is equivalent to pressing the TAB key <count> times.

This command can be used to move either the workspace cursor or the monitor cursor. If the command is typed on the keyboard, it moves the cursor in that scroll which receives text from the keyboard. If the command is sent from the computer, it moves the cursor in that scroll which receives text from the computer.

Suppose text from the computer is printed in the workspace and the command

```
!TAB <count><CR>
```

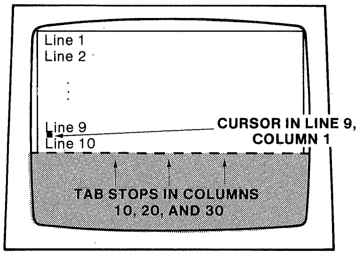
is sent from the computer. This command moves the workspace cursor <count> tab stops to the right. If there are no tab stops defined to the right of the current cursor position, the next tab moves the cursor to the beginning of the next line. Thus if <count> is large enough to move the cursor past the last tab stop in a line, the cursor jumps to column 1 of the next line and continues tabbing a total of <count> stops. Each skip to the next line, as well as each skip to the next tab stop in a line, accounts for one of the <count> tabs. If <count> is large enough to move the cursor below the bottom of the workspace window, roll up occurs.

If <count> is large enough to move the cursor past the last line in the workspace, enough blank lines are created at the bottom of the workspace to accommodate the command.

If the text from the computer is printed in the monitor and this command is sent from the computer, it has the same effect on the monitor cursor.

Examples

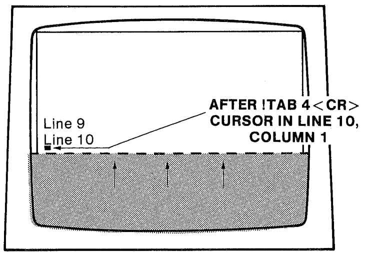
Suppose there is a workspace window of ten lines, with tab stops in columns 10, 20, and 30, and the cursor is in line 9, column 1.


Line 1
Line 2
Line 9
Line 10
CURSOR IN LINE 9, COLUMN 1
TAB STOPS IN COLUMNS 10, 20, AND 30

1. The command

   !TAB 4<CR>

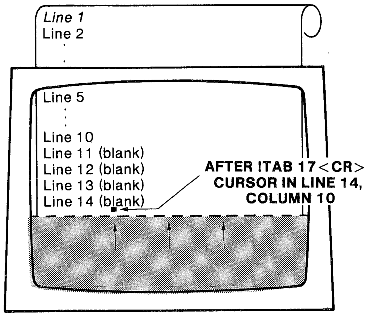
   moves the cursor to the three stops in line 9 and then to column 1 of line 10.



2. The subsequent command

   !TAB 17<CR>

   moves the cursor to column 10 (the first stop) in line 14. The first 16 tabs move the cursor through lines 10, 11, 12, and 13, to column 1 of line 14; the final tab moves the cursor from column 1 of line 14 to the first tab stop in line 14.



NOTE

The TAB command, like the TAB key, performs a different action when the terminal is in form fillout mode. See the Forms and Form Fillout section for details.

BACKTAB

Syntax

```
!BACktab [<count>]<CR>
```

where <count> is a positive integer. If <count> is not specified, it defaults to one.

Action

This command is equivalent to pressing the BACKTAB key (SHIFT-BACKSPACE) <count> times.

This command can be used to move either the workspace cursor or the monitor cursor. If the command is typed on the keyboard, it moves the cursor in that scroll which receives text from the keyboard. If the command is sent from the computer, it moves the cursor in that scroll which receives text from the computer.

Suppose text from the computer is printed in the workspace and the command

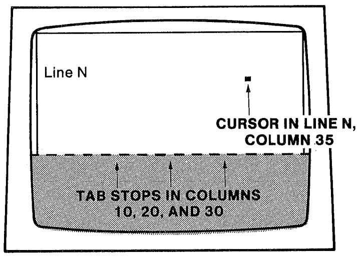
```
!BAC <count><CR>
```

is sent from the computer. This command moves the workspace cursor <count> tab stops to the left. Each backtab moves the cursor one tab stop to the left, or to the left margin if there are no tab stops to the left of the cursor position. The cursor does not move to a preceding line of text, regardless of how large <count> is, but "sticks" at the left margin of the current line.

If text from the computer is printed in the monitor and this command is sent from the computer, it has the same effect on the monitor cursor.

Examples

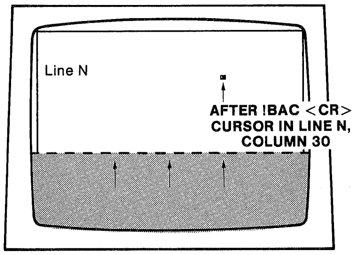
Suppose tab stops are set at columns 10, 20, and 30, and the cursor is in column 35.


Line N
CURSOR IN LINE N, COLUMN 35
TAB STOPS IN COLUMNS 10, 20, AND 30

1. The command

   !BAC<CR>

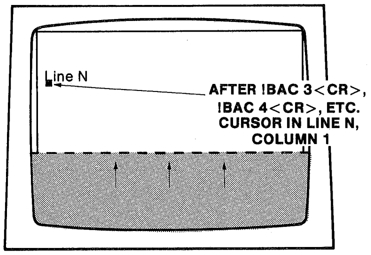
   moves the cursor left one stop to column 30 of the current line.



2. Any of the subsequent commands

   !BAC 3<CR>
   !BAC 4<CR>
   :
   :

   moves the cursor to column 1 of the current line.


Line N
AFTER !BAC 3<CR>, !BAC 4<CR>, ETC. CURSOR IN LINE N, COLUMN 1

NOTE

The BACKTAB command, like the BACKTAB key, performs a different action when the terminal is in form fillout mode. See the Forms and Form Fillout section for details.

THE SCROLLING COMMANDS

RUP (Roll Up)

Syntax

```
!RUP [<count>]<CR>
```

where <count> is a positive integer. If <count> is not specified, it defaults to one.

Action

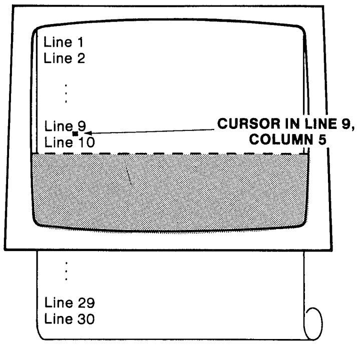
This command is equivalent to pressing the up scrolling key (pad key 7, marked ▲) <count> times.

This command rolls up the current scroll (workspace or monitor) <count> lines, or until the last line of the scroll is visible at the bottom of the window. This command does **not** create blank lines at the end of the scroll. If <count> is larger than the number of lines remaining in the scroll, the scroll rolls up until the last line of the scroll is visible in the window, then stops.

When the scroll rolls up, the cursor moves with it, remaining in the same line of text, at the same column position, as long as that line of text remains visible. If that line of text passes out of the window, the cursor "sticks" at the top of the window, with the column position unchanged.

Examples

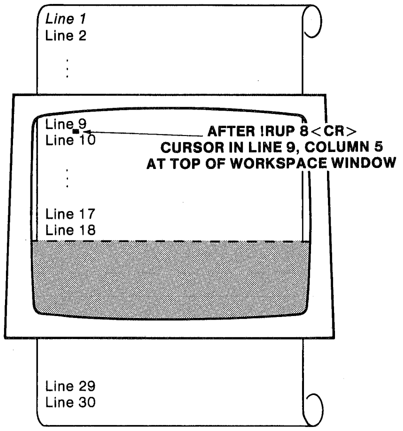
Suppose a workspace window of ten lines is defined, the workspace scroll contains 30 lines, and the cursor is in line 9, column 5.



1. The command

```
!RUP 8<CR>
```

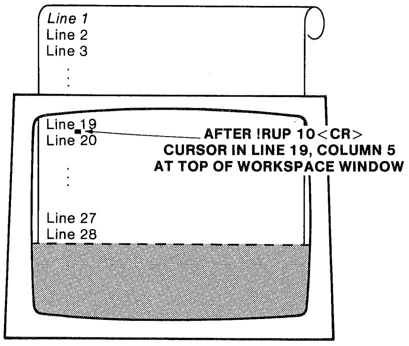
leaves line 9 at the top of the workspace, with the cursor in line 9, column 5.



2. The subsequent command

```
!RUP 10<CR>
```

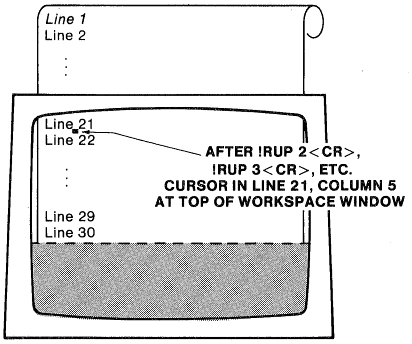
leaves line 19 at the top of the workspace window, with the cursor in line 19, column 5.



3. Any of the subsequent commands

```
!RUP 2<CR>
!RUP 3<CR>
     :
     :
```

leaves line 30 at the bottom of the workspace window, with the cursor in line 21, column 5.



RDOWN

Syntax

```
!RDOwn [<count>]<CR>
```

where <count> is a positive integer. If <count> is not specified, it defaults to one.

Action

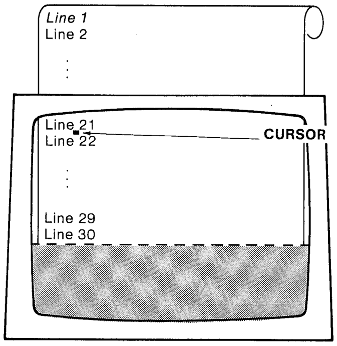
This command rolls down the current scroll (workspace or monitor) <count> lines, or until the first line of the scroll is at the top of the window. The RDOWN command cannot be used to insert blank lines at the top of the workspace.

Giving this command is equivalent to pressing the down scrolling key (pad key 1, marked ▼) <count> times.

When the current scroll rolls down, the cursor moves with it, remaining at the same row and column position as long as that position is visible in the window. If that position passes out of the window, the cursor "sticks" at the bottom line of the window, with the column position remaining unchanged.

Examples

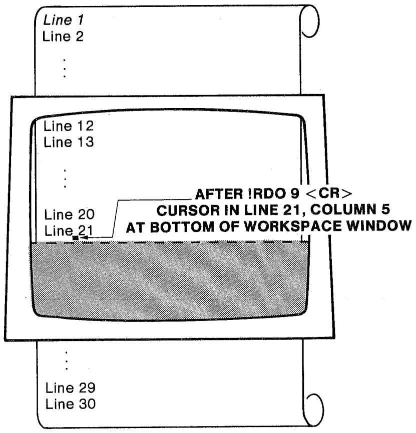
Suppose a workspace window of ten lines is defined, with a workspace scroll of 30 lines and the cursor positioned in line 21, column 5.



1. The command

   !RDO 9<CR>

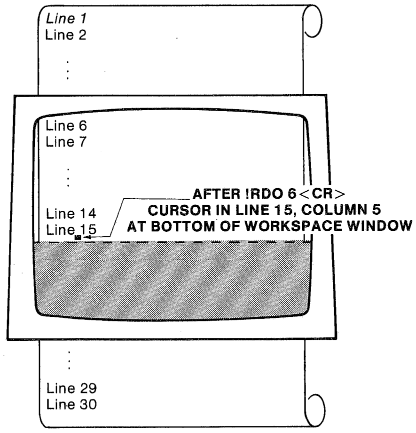
   rolls the workspace down 9 lines, leaving the cursor still positioned in line 21, column 5.



2. The subsequent command

   !RDO 6<CR>

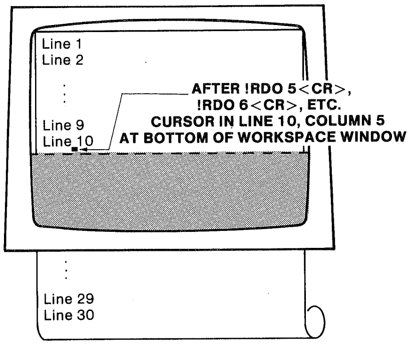
   rolls the workspace down an additional six lines, leaving the cursor in line 15, column 5, at the bottom of the window.



3. Any of the subsequent commands

   !RDO 5<CR>
   !RDO 6<CR>
   !RDO 7<CR>
   :
   :

   rolls the workspace down five lines, with the cursor in line 10, column 5, at the bottom of the window.



ADDITIONAL COMMANDS

ERASE

Syntax

```
!ERAse [ Workspace | Monitor ]< CR>
```

Action

This command erases the specified scroll. The entire scroll, not just the portion visible in the window, is erased. If text is currently directed into that scroll, the cursor quickly reappears in the home position (line 1, column 1, in the upper left corner) of the window. If text is not currently directed into that scroll, the next time that cursor appears, it appears in the home position. This command does not affect the size of the workspace and monitor windows.

If no parameter is specified, the source of the command determines which scroll is erased. If the commmand is sent from the computer and no parameter is specified, the scroll which receives text from the computer is erased. If the command is typed on the keyboard and no parameter is specified, the scroll which receives text from the keyboard is erased.

The ERASE command can also be used to erase the contents of a graphics region in the workspace by entering the command !ERA !ERA G< CR> . See the 4025A Graphics section for details.

Examples

1. !ERA W< CR> — Erases the workspace scroll and returns the workspace cursor to the home position.

2. !ERA M< CR> — Erases the monitor scroll and returns the monitor cursor to the home position.

3. !ERA< CR> — If sent from the computer, this command erases whichever scroll receives text from the computer.

   If typed on the keyboard, this command erases whichever scroll receives text from the keyboard.

BELL

The terminal contains a bell. This bell sounds automatically when certain conditions occur; for example, the bell rings if the operator types beyond the right margin, or if an attempt is made to enter a character in a protected field when the terminal is in form fillout mode.

The programmer may wish to sound the terminal bell at various times during an applications program — perhaps to remind the operator to enter data, or to press a function key. The BELL command is used for this purpose.

Syntax

```
!BELI<CR>
```

Action

The command

```
!BELL<CR>
```

or

```
!BEL<CR>
```

sounds the bell.

The bell also sounds when the ASCII BEL character, CTRL-G, is sent to the terminal.

Section 7

FORMS AND FORM FILLOUT

From the operator's viewpoint, a form consists of several lines of text displayed in the workspace and formatted in a particular way. A form is divided into blanks areas, which the operator fills in, and labels, which identify the type of data to be entered in each blank. There may also be horizontal and vertical ruling lines to emphasize the structure of the form. The operator fills in the blanks with appropriate data and sends this data to the computer for storage or processing.

A sample form used to store a customer's name and address is shown in Figure 7-1, with the blanks shaded gray.

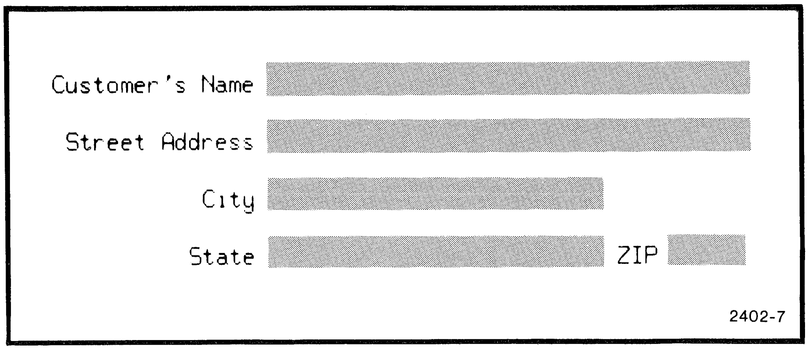


Figure 7-1. Sample Form.

FORM FILLOUT MODE

A form is filled out and the data in the form sent to the computer while the terminal is in form fillout mode.

Form fillout mode has several features designed to make it easy to fill out and process forms.

- Data can be entered only in the blanks of the form. These blanks are called **unprotected fields**. If the operator attempts to enter a character elsewhere in the form (in a **protected field**), the terminal bell sounds and the character is inserted in the next unprotected field in the form.

- Several keys on the keyboard behave differently when the keyboard types into the workspace. The TAB key moves the cursor to the beginning of the next unprotected field of the form. The BK TAB key moves the cursor to the beginning of the preceding unprotected field. The HOME key moves the cursor to the beginning of the first unprotected field in the form, rather than to column 1, row 1 of the workspace. The ERASE key erases only the data in the unprotected fields; protected fields are not erased.

- Several commands have effects other than the usual ones. When the computer types into the workspace, the TAB, BACKTAB, and ERASE commands have the same effects as the corresponding keys. The editing commands also behave differently. These differences are detailed, command by command, throughout this section and later sections.

A typical form fillout application includes the following steps:

- Insure that the terminal is not in form fillout mode.

- Display the form in the workspace. Either the operator creates the form from the keyboard or, more usually, a stored form is sent from the computer or tape unit to the workspace. Both processes are the same from the terminal's viewpoint.

- Place the terminal in form fillout mode.

- Fill out the form.

- Send the data in the form to the computer, printer, a tape unit, or a hard copy unit.

- Erase the unprotected fields of form and fill it out again; then send the new data in the form to the computer. Repeat this procedure as long as necessary.

- When the form is no longer needed, remove the terminal from form fillout mode and erase the form itself from the screen.

The FORM command is used to place the terminal in form fillout mode and to remove it from form fillout mode.

FORM

Syntax

```
!FORm [ Yes|No ]<CR>
```

If no parameter is specified, Yes is assumed.

Action

If Yes is specified, the terminal is placed in form fillout mode. If No is specified, the terminal is removed from form fillout mode.

Examples

!FOR<CR> !FOR Y<CR> !FORM YES<CR>	Places the terminal in form fillout mode.
!FOR N<CR> !FORM NO<CR>	Removes the terminal from form fillout mode.

CREATING A FORM

From the terminal's viewpoint, there is more to a form than meets the eye. Consider the sample form in Figure 7-2. This form, and every form, consists of several lines of text. Each line is divided into one or more sections called fields; each field is divided into individual character positions.

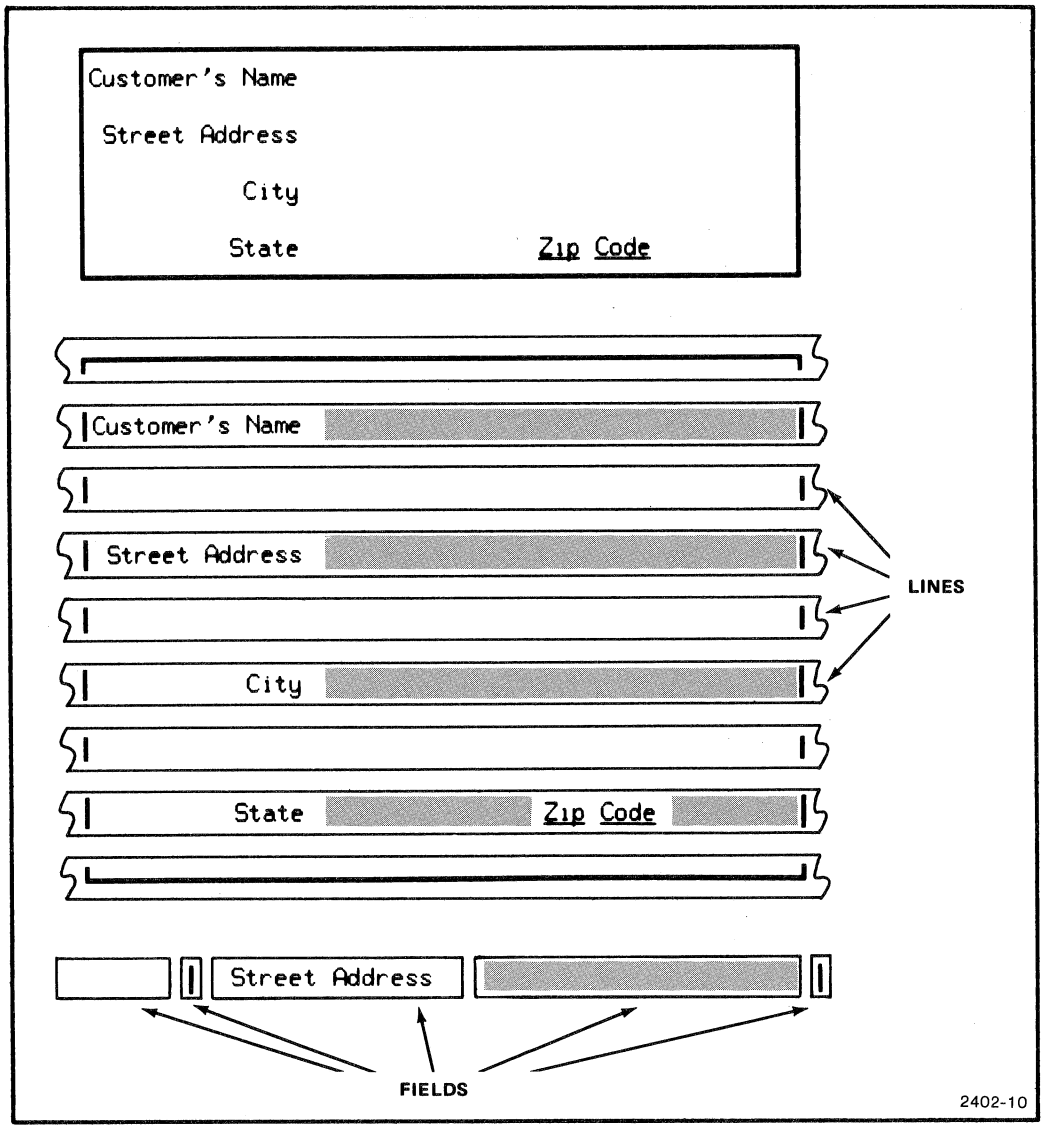


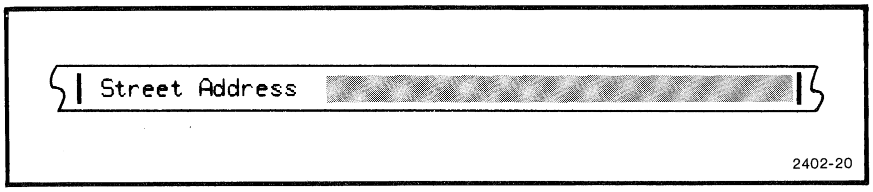
Figure 7-2. The Parts of a Form.

To display a form, the terminal stores the information which defines the form in the portion of memory called the workspace display list. In addition to the characters which are displayed on the screen, the display list includes markers which are not displayed. These markers are of two types:

- End-of-line markers which indicate where one line of text ends and the next begins.

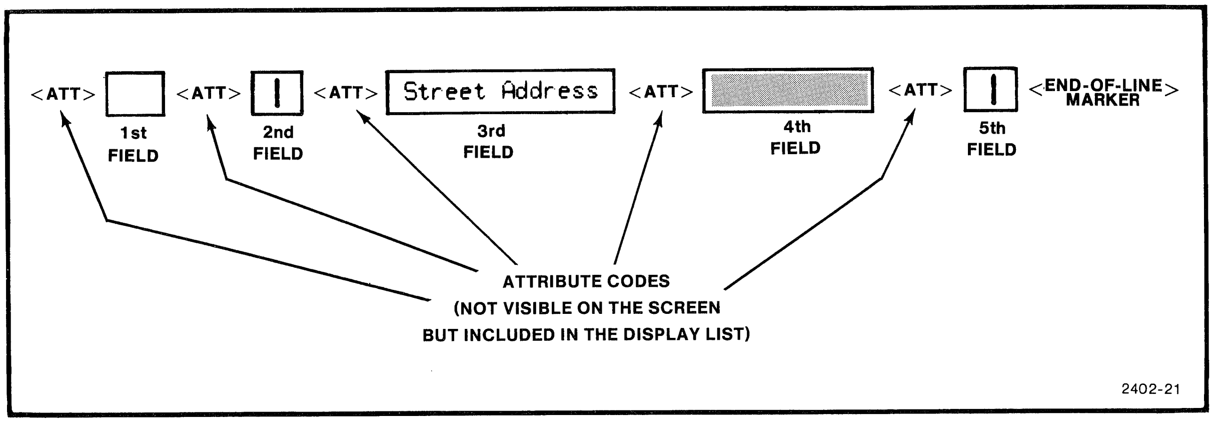
- Markers called attribute codes. Attribute codes divide a line into fields and determine the properties, or attributes, of those fields.

The fourth line of the sample form appears on the screen as follows:



2402-20

In the workspace display list, however, the following information is stored:



2402-21

To create a form one must complete the following steps:

- Decide what each line of the form is to look like — what attributes each field will possess; what text, if any, will be printed in the protected fields.

- Attach attribute codes to each field so that when the terminal displays the form, each field will have the desired attributes and the form, as a whole, will have the desired appearance.

FIELD ATTRIBUTES AND FIELD ATTRIBUTE CODES

There are three classes of field attributes:

- Character font attributes: font zero, font one, font two, etc.
- Logical attributes: alphanumeric, numeric, protected, and protected modified. Alphanumeric and numeric denote unprotected fields into which the operator can enter data.
- Visual attributes: standard, enhanced, inverted, underscored, and combinations of these.

Font Attributes

A font attribute is an integer between 0 and 31, inclusive. The integer designates the character font from which characters are selected for display in the field. The default font attribute is 0. Font 0 is called the standard font and consists of the 128 characters of the ASCII code. Font 1 is always the Ruling Characters font (Option 32); Font 2 is the Math Characters font (Option 34). On the 4025A, other fonts may be determined by ROMs inserted in the Character Set Expansion Board (Option 31), or may be defined by the user with SYMBOL commands (if the terminal contains Option 25 or 26).

If a font attribute is specified for which no character font is defined, each character in the font is displayed as a rectangle with all the dots in that character cell matrix turned on.

NOTE

Font attributes in the display list affect the display whether the terminal is in form fillout mode or not. A field with font attribute 1, for example, displays characters from Font 1 at all times (assuming Option 32 is present).

Logical Attributes

The logical attributes which a field can possess are as follows:

Symbol Used	Attribute	Meaning
A	Alphanumeric	The default logical attribute. Specifies an alphanumeric unprotected field into which any alphanumeric character may be entered.
N	Numeric	Specifies a numeric unprotected field. In form fillout mode, only characters with ADEs 32-63 can be entered in a numeric field. (This includes the numerals 0-9, and most punctuation symbols.)
P	Protected	Specifies a protected field. In form fillout mode, a protected field cannot be typed into or erased.

Note that the fields of the form to be filled in by the operator must be unprotected fields, with logical attributes A or N; labels and areas in which the operator is not to type should be protected.

Each field possesses one of these logical attributes. In addition, any field may possess the logical attribute M, for "modified." The SEND MOD command sends to the computer the data in those, and only those, fields which have been flagged as "modified" with the logical attribute M. (See the discussion of the SEND command later in this section.)

A field may be flagged as "modified" in either of two ways:

- When the data in any unprotected field is changed, the terminal automatically attaches the logical attribute M to that field. The next SEND MOD command sends the data in that field to the computer and removes the M attribute. The data in this field is not sent to the computer again until it has been modified in some way and the field once again flagged with the logical attribute M.

- The ATTRIBUTE command may specify the logical attribute PM, for "protected modified." A SEND MOD command sends the data in such a field to the computer, but does not remove the M attribute; thus a PM field is sent to the computer with every SEND MOD command.

NOTE

Logical attributes have effect only when the terminal is in form fillout mode. When not in form fillout mode, the terminal ignores logical attributes.

Visual Attributes

The visual attributes which a field can possess are as follows:

Symbol Used	Attribute	Meaning
S	Standard	Displays light characters on a dark background. This is the default visual attribute.
E	Enhanced	Displays bright characters on a light background. (The absolute brightness and contrast are controlled manually by the operator.)
I	Inverted	Displays dark characters on a light background.
U	Underscored	Underscores all characters and spaces in the field.

A field can possess two or three visual attributes simultaneously. If a field possesses visual attributes of EU, characters in the field are displayed both enhanced and underscored.

NOTE

Like font attributes, visual attributes affect the display even when the terminal is not in form fillout mode.

Field Attribute Codes Within a Line

Unless instructed otherwise by an ATTRIBUTE command, the terminal begins each line with the default attribute in each class: font 0, alphanumeric logical attribute, and standard visual attribute.

An attribute code may specify attributes from one, two or all three classes of field attributes. As the terminal scans each line in its display list, it searches for attribute codes. When it encounters a new attribute code, it modifies only the class or classes of attributes specified in this new code; the other class or classes of attributes are not modified. Suppose, for example, the following line is stored in the display list:

< font 0,protected,enhanced> -----------------< numeric> ----------< CR>

Since the second attribute code specifies only the logical attribute numeric, the second field is displayed in font 0, enhanced (the font and visual attributes of the preceding field).

CREATING FIELDS

Each field in a line is created by specifying the font, logical, and visual attributes which the field possesses. The ATTRIBUTE command is used for this purpose.

ATTRIBUTE

Syntax

```
!ATTribute [<font>] [<logical>] [<visual> [—<visual>] ]<CR>
```

where <font> denotes a font attribute, <logical> denotes a logical attribute, and each <visual> denotes one or more visual attributes.

Action

The ATTRIBUTE command inserts a field attribute code into the workspace display list at the cursor position. This field attribute code marks the beginning of a new field and designates the font, logical, and/or visual attributes of this field, as specified in the ATTRIBUTE command. If this field is the first field in the line, the ATTRIBUTE command specifies the attributes of the field which differ from the default attributes. If the field is preceded by another field on the same line, the ATTRIBUTE command specifies the attributes of the new field which differ from those of the preceding field. If two visual attributes or sets of attributes are separated by a hyphen, the display blinks that field between the two specified attributes or sets of attributes.

Restrictions on Syntax

The <font> is an integer between 0 and 31, inclusive. The <font> defaults to 0 (at the beginning of a line) or to the font attribute of the preceding field.

```
<logical> = [ A|N|P|PM ]
```

where A denotes alphanumeric, N denotes numeric, P denotes protected, and PM denotes protected modified. These parameters must be given in this single letter form.

The <logical> defaults to A (at the beginning of a line) or to the logical attribute of the preceding field.

<visual> = [S | E | I | U | EI | EU | IU | EIU]

and

—<visual> = —[S | E | I | U | EI | EU | IU | EIU]

where S denotes Standard, E denotes enhanced, I denotes inverted, and U denotes underscored. If more than one letter is specified, the order of the letters does not matter; EI and IE give the same visual characteristics to the field.

If the —<visual> parameter is specified, the display blinks between the two attributes or sets of attributes specified. For example, visual attributes of E-I cause the field to blink between enhanced and inverted visual attributes.

The <visual> defaults to S (at the beginning of a line) or to the visual attribute(s) of the preceding field.

No spaces are allowed between alphabetic parameters in the ATTRIBUTE command. To define a protected field with the enhanced and inverted visual attributes, for example, give the command

```
!ATT PEI<CR>
```

To blink that field between the enhanced and inverted visual attributes, give the command

```
!ATT PE-I<CR>
```

Examples of ATTRIBUTE Commands

Font Attributes

!ATT<CR>
!ATT 0

Defines a new field beginning at the cursor position. When characters are entered in this field, the field displays characters from Font 0, the standard font.

!ATT 1<CR>

Defines a new field beginning at the cursor position. When characters are entered in this field, the field displays the corresponding characters from Font 1, the Ruling Characters font. (Requires Option 32)

Logical Attributes

!ATT A<CR>

Defines an alphanumeric unprotected field beginning at the cursor position.

!ATT N<CR>

Defines a numeric unprotected field beginning at the cursor position. In form fillout mode, only characters with ADEs 32-63 can appear in this field.

!ATT P<CR>

Defines a protected field, beginning at the cursor position. In form fillout mode, this field cannot be typed into or erased.

!ATT PM<CR>

Defines a protected modified field beginning at the cursor position. This field is transmitted to the computer with each subsequent SEND MOD command.

Visual Attributes

!ATT S<CR>

Defines a new field beginning at the cursor position. Displays that field with the standard visual attribute — light characters on dark background.

!ATT E<CR>

Defines a new field beginning at the cursor position. Displays that field with the enhanced visual attribute — bright characters on light background.

!ATT I<CR>

Defines a new field beginning at the cursor position. Displays that field with the inverted visual attribute — dark characters on light background.

!ATT U<CR>

Defines a new field beginning at the cursor position. Displays that field with the underscored visual attribute — all characters and spaces in the field are underscored.

!ATT E-S<CR>

Defines a new field beginning at the cursor position and blinks that field between the visual attributes of enhanced and standard.

!ATT I-EU<CR>

Defines a new field beginning at the cursor position and blinks that field between the visual attributes of inverted and enhanced with underscore.

Combined Attributes (No spaces allowed between alphabetic parameters)

!ATT AEU<CR>

Defines an alphanumeric field beginning at the cursor position. Displays that field with the enhanced and underscored visual attributes.

!ATT NI-EU<CR>

Defines a numeric field beginning at the cursor position. Blinks that field between the visual attributes of inverted and enhanced with underscore.

!ATT 1 PS-E<CR>

Defines a protected field beginning at the cursor position. Blinks that field between the visual attributes of standard and enhanced. When characters are entered in this field, the field displays characters from Font 1 (Ruling Characters). (Requires Option 32)

Creating Fields with JUMP

The JUMP command can be used with the ATTRIBUTE command to create several fields on one line. Suppose you want to create a protected enhanced field 60 character positions in length in row 3 of the workspace. The command

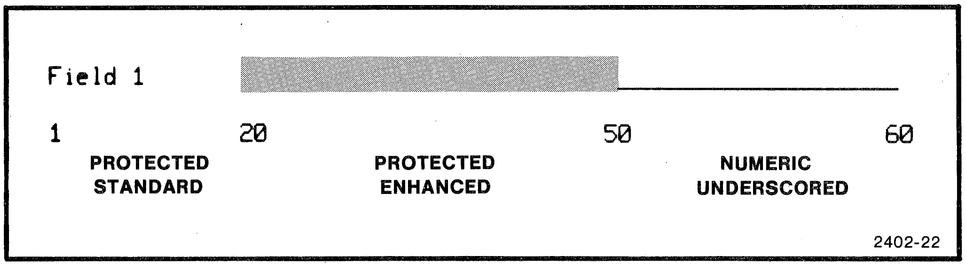
```
!JUM 3!ATT PE;————————(60 spaces)———————<CR>
```

creates the desired field. However, the command

```
!JUM 3!ATT PE!JUM 3,60!ATT PS<CR>
```

creates the desired field more quickly and with more efficient coding.

The JUMP command can be used to quickly and efficiently create several fields on one line of the workspace. Suppose you want row 5 to appear as follows:



2402-22

This can be done by transmitting the fields as series of spaces, as in our first example. But the command sequence

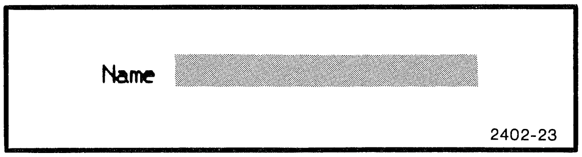
```
!JUM 5!ATT PS;Field 1!JUM 5,20!ATT E!JUM 5,50!ATT NE-IU;Field 3———<CR>
```

gives the same display and transmits 21 fewer characters than the first method.

Suppose the workspace cursor is in the home position (row 1, column 1) and consider the three command sequences:

1. !ATT P;Name !ATT AE!JUM 1,25!ATT PS<CR>
2. !ATT P;Name !ATT AE!JUM 1,25!ATT S!JUM 1,60!ATT PS<CR>
3. !ATT P:Name !ATT AE!JUM 1,25!ATT PS!JUM 1,80!ATT PS<CR>

When executed, each of these command sequences causes the same display:



Each sequence, however, creates a very different "line" in the terminal display list, and the differences between them are important when the terminal is in form fillout mode.

The line generated by 1. ends in column 25; the display list contains nothing beyond that column. If the operator moves the cursor right of column 25 in line 1 and presses a key, the cursor moves to the beginning of the next unprotected field and prints the typed character there. The terminal bell does not ring.

The line generated by 2. ends in column 60. Columns 26 through 60 constitute an unprotected field. If the operator types in these columns, the text is printed just as it is typed.

The line generated by 3. ends in column 80; all 80 columns of the screen are included in this line. Columns 26 through 79 constitute a protected field. If the operator moves the cursor into this field and types a character, the terminal bell rings, the cursor moves to the beginning of the next unprotected field in the form, and the character is printed there.

CAUTION

When using the Jump command to create fields, always "tie down" the line with the !ATT PS command, as shown in the preceding examples. If this is not done, the display list may not include the last field created with JUMP.

RULINGS

One can highlight the structure of a form by drawing rulings, or ruling lines. The terminal with the Ruling Characters font (Option 32) has two provisions for doing this. First, the basic command set includes the HRULE (Horizontal Rule) and VRULE (Vertical Rule) commands. Second, the Ruling Characters font (Option 32) itself provides additional ruling characters for making junctions between horizontal and vertical rulings.

HRULE

(Requires Option 32)

Syntax

```
!HRUle <row> <column> [<length> [<width>] ]<CR>
```

where all parameters are positive integers. The <row> and <column> parameters give absolute workspace coordinates (as in the JUMP command). Since there are only 80 columns, the <column> parameter must not exceed 80 and the sum of <column> and <length> must not exceed 81. The <width> parameter, if specified, must be either 1 or 2. The default value for both <length> and <width> is 1.

Action

This command draws a horizontal ruling in the workspace. The first character of the ruling is inserted at the row and column specified by the <row> and <columnp> parameters. The ruling continues to the right for a total of <length> columns. This ruling is a single line if <width> is 1 and a double line if <width> is 2.

Examples

!HRU 3,5,20<CR> !HRU 3,5,20,1<CR>	Beginning at row 3, column 5 of the workspace, draws a horizontal ruling through 20 columns (columns 5 through 24). The ruling is a single line.
!HRU 3,5,20,2<CR>	Beginning at row 3, column 5 of the workspace, draws a horizontal ruling through 20 columns (columns 5 through 24). The ruling is a double line.

VRULE

(Requires Option 32)

Syntax

```
!VRUle <row> <column> [<length> [<width>]]<CR>
```

where all parameters are positive integers. The <row> and <column> parameters are absolute workspace coordinates (as in the JUMP command). The <column> parameter may not exceed 80. The <width> parameter, if specified, must be either 1 or 2. The default value of both <length> and <width> is 1.

Action

This command draws a vertical ruling in the workspace. The first ruling character is inserted at the row and column specified by the <row> and <column> parameters. The ruling continues downward for a total of <length> rows. If <width> is 1 (or omitted), the ruling is a single line; if <width> is 2, the ruling is a double line.

Examples

!VRU 3,5,20<CR> !VRU 3,5,20,1<CR>	Beginning at row 3, column 5 of the workspace, draws a vertical ruling through 20 rows (rows 3 through 22). This ruling is a single line.
!VRU 3,5,20,2<CR>	Beginning at row 3, column 5 of the workspace, draws a vertical ruling through 20 rows (rows 3 through 22). This ruling is a double line.

NOTE

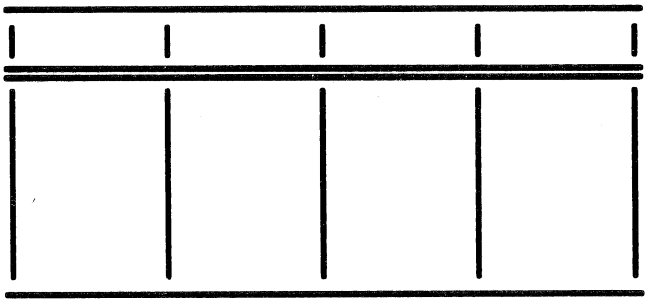
If the terminal receives an HRULE or VRULE command but does not contain Option 32, each character cell affected by the command is displayed as a bright rectangle with all its matrix dots turned on. This also happens if an ATTRIBUTE command <font> specifies a character font for which no ROM is installed or no character definitions have been given. In this way, the HRULE and VRULE commands may still be used to highlight the structure of a form.

Making Correct Junctions

While the HRULE and VRULE command are convenient, vertical and horizontal rulings drawn with these commands do not cross or join each other. Each ruling character occupies an entire character cell on the display, and a character cell which contains a vertical ruling character cannot contain a horizontal ruling character. For example, suppose we give the following sequence of commands:

```
!VRU 3,20,10,1<CR>
!VRU 3,30,10,1<CR>
!VRU 3,40,10,1<CR>
!VRU 3,50,10,1<CR>
!VRU 3,60,10,1<CR>
!HRU 3,20,41,1<CR>
!HRU 5,20,41,2<CR>
!HRU 12,20,41,1<CR>
```

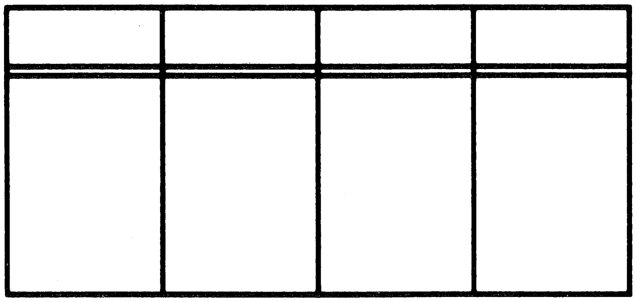
At this point, the basic structure of the form has been created, but the junctions between horizontal and vertical rulings need to be added. The workspace display appears as follows:



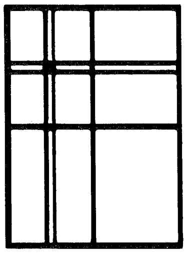
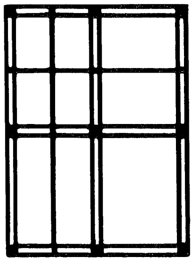
The variety of ruling characters provided in the Ruling Characters Font (Option 32) allows the programmer or operator to make neat, well-fitted junctions by selecting appropriate font characters. The Ruling Junctions Chart (Figure 7-3) is a reference sheet for making junctions. You can make junctions for our sample form with the sequence of commands:

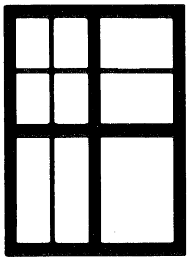
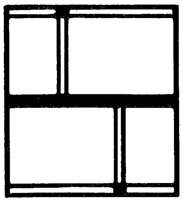
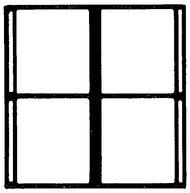
```
!JUM3,20!ATT1; @ !JUM3,30;A!JUM3,40;A!JUM3,50;A!JUM3,60;B<CR>
!JUM5,20!ATT1;!JUM5,30;K!JUM5,40;K!JUM5,50;K!JUM5,60; ∧ <CR>
!JUM12,20!ATT1;P!JUM12,30;Q!JUM12,40,Q!JUM12,50;Q!JUM12,60;R<CR>
```

A complete table of font characters is given in Appendix D.



Now the form looks like this:

RULINGS	
Rulings (Font 1)	**Standard (Font 0)**
	@YYGYYAYYYYYB [_ [[\]]M]]K]]]]]^ [_ [[HYYOYYIYYYYYJ [_ [[[_ [[[_ [[PYYWYYQYYYYYR
	D]]C]]E]]]]]F _ [_ _ XYYIYYOYYYYYZ _ [_ _ L]]K]]M]]]]]N _ [_ _ _ [_ _ _ [_ _ T]]S]]U]]]]]V

RULINGS	
Rulings (Font 1)	**Standard (Font 0)**
	d))c))e)))))f ? [? ? xYYIYYoYYYYYz ? [? ? l))k))m)))))n ? [? ? ? [? ? ? [? ? t))s))u)))))v
	h]]]E]]]]]]]j [_ [[_ [:)))y)))()))~ [_ [[_ [p]]]]]]]U]]]r
	a]]]]]g]]]]]b _ ? _ _ ? _ XYYYYYoYYYYYX _ ? _ _ ? _ qYYYYYwYYYYYi

(2402) 4172-4

Figure 7-3. Ruling Junctions Chart.

THE EFFECT OF FORM FILLOUT ON COMMANDS AND KEYS

Form fillout mode alters the action of some commands and keys, but does not affect the action of others. For commands discussed in later sections, any effects of form fillout mode on a command are discussed when the command is introduced. Some of the display control commands already discussed are affected by form fillout mode:

- The TAB, BACKTAB, and ERASE commands (and their corresponding keys) are affected by form fillout mode.
- The UP, DOWN, RIGHT, LEFT, RUP, and RDOWN commands (and their corresponding keys) are not affected by form fillout mode. The JUMP command is not affected by form fillout mode, but is still useful for working with forms.

The following discussion assumes that the terminal is in form fillout mode, that commands come from the computer, and that text from the computer is directed into the workspace.

Typing in Form Fillout

When the terminal is in form fillout mode, text can be entered only in the unprotected fields of the form. If the operator types a character while the workspace cursor is in a protected field, the terminal bell rings and the typed character is inserted in the first column of the next unprotected field in the form.

If the cursor is in the last column of an unprotected field and the operator types a character, the character is inserted in that column and the cursor moves to the first column of the next unprotected field of the form.

If the cursor is moved beyond the last field in a line (using JUMP or a cursor key) and a character is typed, the cursor moves to the beginning of the next unprotected field in the form and the typed character is entered there. In this case, the terminal bell does not ring. (See the Creating Fields with JUMP discussion earlier in this section.)

When a form is created, a line of the form may consist only of a <CR>. Such a line contains no protected or unprotected fields; it appears on the terminal screen as a blank line, but in the workspace display list only a <CR> is stored. If the cursor is positioned anywhere in such a line, it is (strictly speaking) beyond the end of the line. If a character is typed when the cursor is in such a line, the cursor moves to the beginning of the next unprotected field in the form; the terminal bell does not ring.

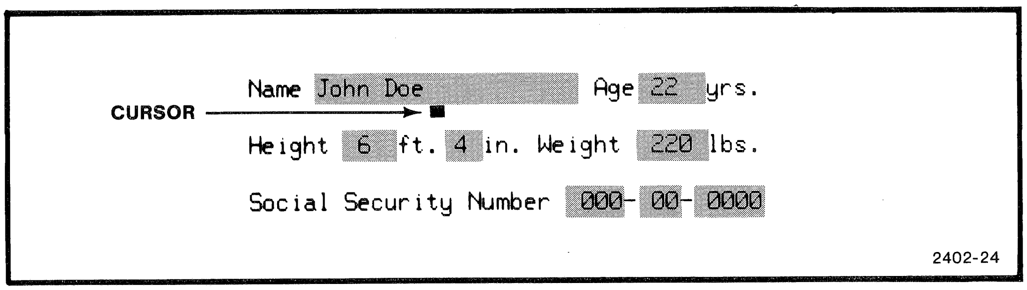
If the cursor is moved beyond the last unprotected field in the form and a key is pressed, the cursor moves to column 1 of the last line in the workspace window. The typed character is not displayed.

TAB in Form Fillout

Each tab character advances the workspace cursor to the beginning of the next unprotected field in the form. If the cursor is in the last unprotected field of the form, the next tab character sends the cursor to the home position at the beginning of the first unprotected field.

Examples

Suppose the sample form shown at right is displayed in the workspace, with the cursor positioned as shown.



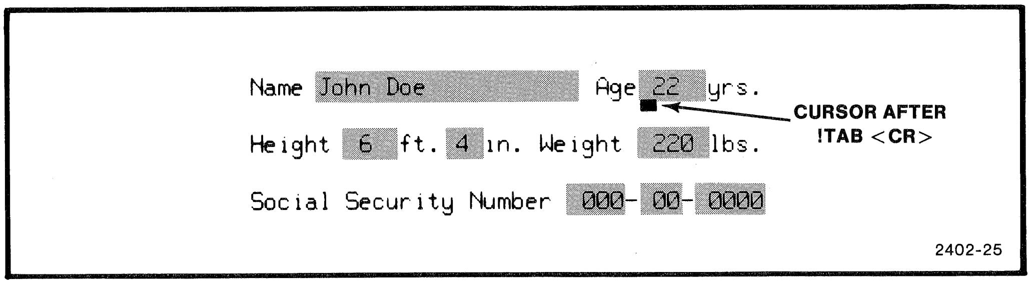
```
Name John Doe            Age 22 yrs.
Height  6 ft. 4 in. Weight 220 lbs.
Social Security Number 000- 00- 0000
```

2402-24

1. The command

   !TAB<CR>

   moves the cursor to the beginning of the next unprotected field.



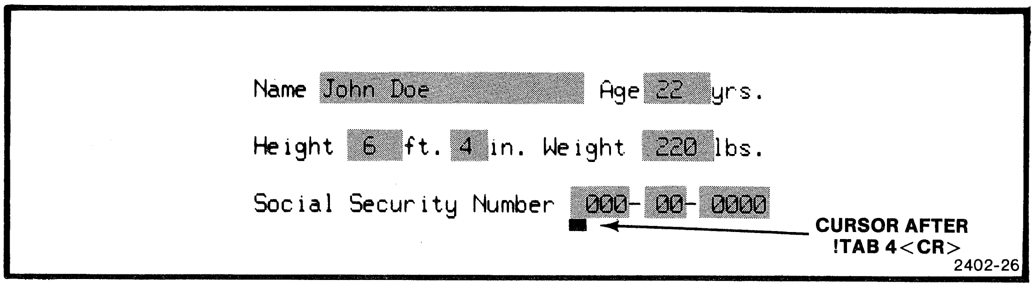
```
Name John Doe            Age 22 yrs.
Height  6 ft. 4 in. Weight 220 lbs.
Social Security Number 000- 00- 0000
```

2402-25

2. The subsequent command

   !TAB 4<CR>

   advances the cursor four unprotected fields and positions it as shown.



```
Name John Doe            Age 22 yrs.
Height  6 ft. 4 in. Weight 220 lbs.
Social Security Number 000-00-0000
```

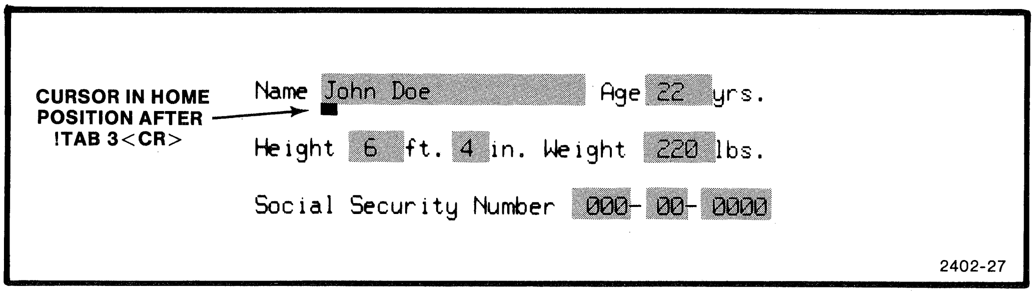
CURSOR AFTER !TAB 4<CR>

2402-26

3. The subsequent command

   !TAB 3<CR>

   advances the cursor through the last two unprotected fields of the form and back to the home position.



```
Name John Doe            Age 22 yrs.
Height  6 ft. 4 in. Weight 220 lbs.
Social Security Number 000-00-0000
```

CURSOR IN HOME POSITION AFTER !TAB 3<CR>

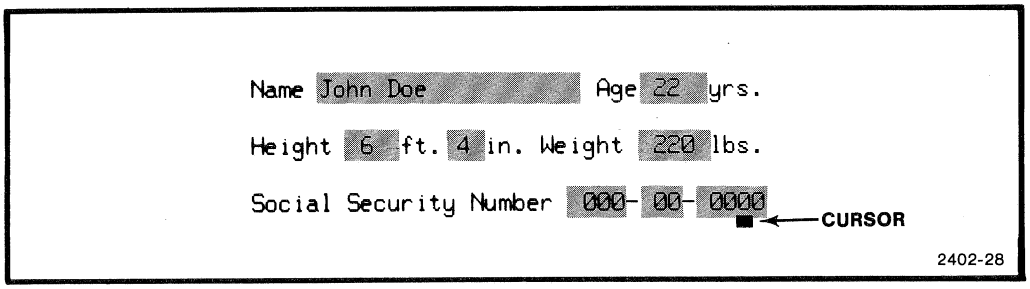
2402-27

BACKTAB in Form Fillout

A BACKTAB character moves the cursor to the beginning of the unprotected field in which it is located. If the cursor is already at the start of an unprotected field, or if it is not inside an unprotected field, a BACKTAB character moves the cursor to the start of the preceding unprotected field. If the cursor is already at the start of the first unprotected field in the form, a BACKTAB character leaves the cursor where it is.

Examples

Suppose the cursor is positioned in the last unprotected field of our sample form, as shown.



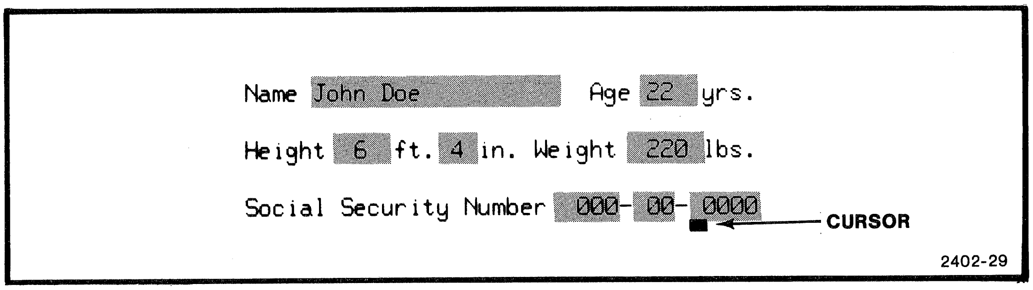
```
Name John Doe             Age 22  yrs.

Height  6  ft. 4 in. Weight  220 lbs.

Social Security Number  000- 00- 0000
```

2402-28

1. The command

   !BAC<CR>

   moves the cursor to the beginning of the unprotected field in which it is located.



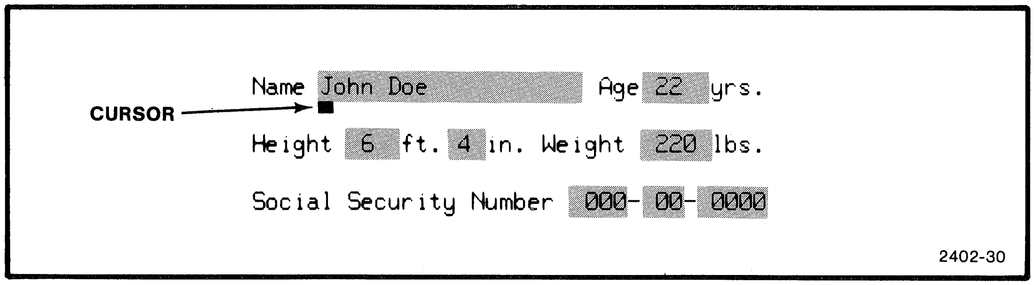
```
Name John Doe             Age 22  yrs.

Height  6  ft. 4 in. Weight  220 lbs.

Social Security Number  000- 00- 0000
```

2402-29

2. Any of the subsequent commands

```
!BAC 7<CR>
!BAC 8<CR>
!BAC 9<CR>
     :
     :
```

moves the cursor through all the preceding seven fields of the form, to the beginning of the first unprotected field.



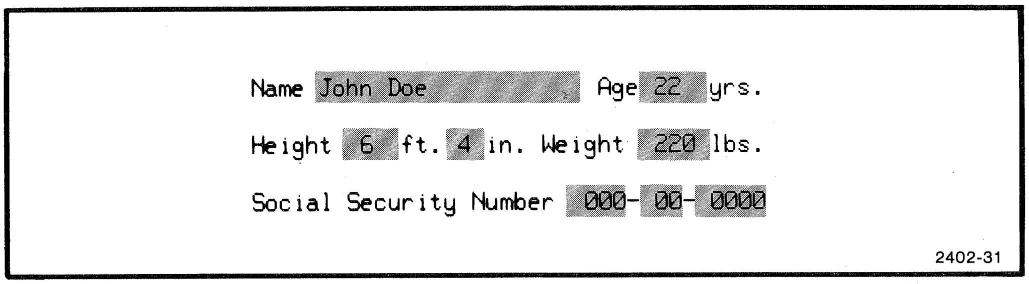
```
Name John Doe              Age 22 yrs.
■
Height  6  ft.  4 in. Weight  220 lbs.

Social Security Number  000- 00- 0000
```

ERASE in Form Fillout

In form fillout mode, the ERASE command erases only the contents of the unprotected fields in the form and leaves the cursor at the beginning of the first unprotected field.

Example

Suppose a sample form is filled out as shown and the information in the form is sent to the computer.



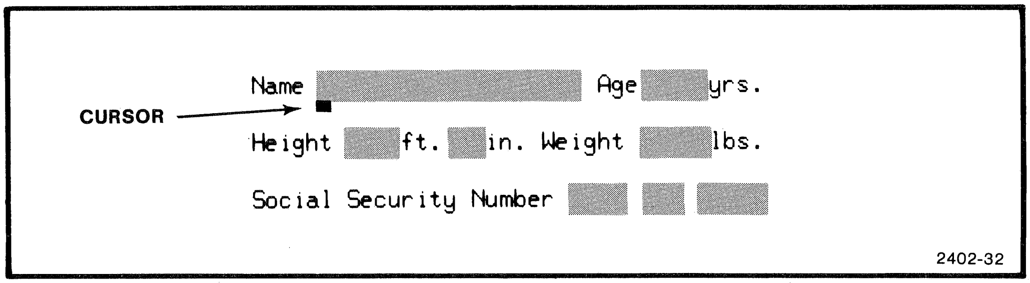
```
Name John Doe           Age 22 yrs.

Height  6 ft. 4 in. Weight 220 lbs.

Social Security Number  000- 00- 0000
```

2402-31

The command

!ERA W<CR>

erases the contents of the form and leaves the cursor positioned as shown.



```
Name                    Age    yrs.

Height     ft.   in. Weight     lbs.

Social Security Number
```

2402-32

The command

!ERA<CR>

also does this if, as we assume here, the device issuing the command (computer or keyboard) also types into the workspace.

The HOME Key and JUMP in Form Fillout

When the terminal is *not* in form fillout mode, the command

```
!JUM<CR>
```

has the same effect as pressing the HOME key. In each case, the workspace cursor moves to row 1, column 1, the "home" position.

In form fillout mode, pressing the HOME key moves the cursor to the beginning of the first unprotected field in the form, which does not necessarily begin in row 1, column 1. But the JUMP command has no respect for form fillout mode. Giving the command

```
!JUM<CR> or
!JUM 1,1<CR>
```

moves the cursor to row 1, column 1, whether that field is protected, unprotected, or even part of the form.

The JUMP command can still be used, with the TAB command, to simulate the action of the HOME key. As long as row 1, column 1 is a protected location, the sequence of commands

```
!JUM!TAB<CR>
```

moves the cursor first to row 1, column 1 (!JUM), then to the beginning of the first unprotected field in the form (!TAB).

TRANSMITTING FORMS AND FORM DATA

Because of the formatted nature of forms and form data, special care must be taken when transmitting either to the computer. The SEND command and the FIELD command have been specially designed for transmitting form information.

SEND in Form Fillout

Syntax

```
!SENd [ All | Mod ]<CR>
```

The default parameter is All; that is, !SEN<CR> is equivalent to !SEN A<CR>.

There are two uses of the SEND command involving forms.

First, suppose the operator has constructed a form in the workspace and wishes to store this form in the computer. (The terminal with Option 04 can also store forms or form data on a TEKTRONIX 4924 Digital Cartridge Tape Drive. See the Peripherals section.) After making sure that the terminal is not in form fillout mode, the operator gives the SEND command. (If the terminal is not in form fillout mode, the SEND, SEND ALL, and SEND MOD commands are equivalent.) This command sends all the information in the workspace to the computer. Field attribute codes in the workspace display list are automatically encoded as ATTRIBUTE commands; thus, when the form is sent back to the terminal from the computer, the terminal has the necessary information to reconstruct the form.

Second, suppose a form is displayed, the terminal is placed in form fillout mode, and the form is filled out. The operator now wishes to send the data in the form (not the form itself) to the computer for storage or processing. With the terminal in form fillout mode, the operator uses either the SEND ALL command or the SEND MOD command.

The SEND ALL command sends to the computer the data in each unprotected field of the form.

The SEND MOD command sends to the computer the data in just those fields flagged with the logical attribute M (modified). In this case, the data in a field is sent to the computer if and only if (1) the field is an unprotected field whose contents have been changed since the last SEND or SEND MOD command, or (2) the field is a protected field permanently flagged with the logical attribute PM (protected modified).

When a SEND MOD command is given, then, consecutive blocks of data received by the computer may not come from consecutive unprotected fields in the form. For the applications program to process the data correctly, however, it must know the form location from which each block of data comes. Therefore, when a SEND MOD command is executed, the data from each modified field is sent to the computer, preceded by a pair of three-digit numbers separated by a comma. These numbers specify, in absolute workspace coordinates, the row (first number) and column (second number) of the first character position of the field. Suppose, for example, a modified field begins in row 5, column 3. When the data in this field is sent to the computer, it is preceded by the string 005,003. Examples of transmissions using the SEND MOD command appear later in this section.

The commands or keyboard operations which affect the contents of a form are ERASE WORKSPACE, DELETE CHARACTER, DELETE LINE, ERASE & SKIP, or the additions and changes made to the unprotected fields.

The source of the !ERA W, !DCH, or !DLI commands, as well as any updates to the contents of the form, determines the affect on the logical attribute modified flag. If these commands or updates are given from the host computer, the fields are NOT flagged as modified. Any keyboard operation, however, which affects the status of unprotected fields, flags those fields to be transmitted on the next SEND MODIFIED command.

In the case of the keyboard ERASE of the workspace, or the !ERA W command from the keyboard, all unprotected fields are marked as modified and are transmitted with the row and column position on the next SEND MODIFIED. This informs the host that information has been removed from the screen.

FIELD in Form Fillout

Syntax

```
!FIEld [<separator>]<CR>
```

where <separator> is a single printing ASCII character or a two- or three-digit ADE of an ASCII character. If no <separator> is specified, it is assumed to be NUL, whose ADE is 00.

Action

The FIELD command sets the field separator. If any non-NUL field separator is specified, that character precedes the data sent to the computer from each field; trailing spaces are not transmitted.

If no field separator (or the NUL separator) is specified, all the data in each field is transmitted to the computer by a SEND (ALL or MOD) command. If a field is not completely filled out, all the spaces at the end of the field are treated as data and sent to the computer, along with the rest of the data in the field.

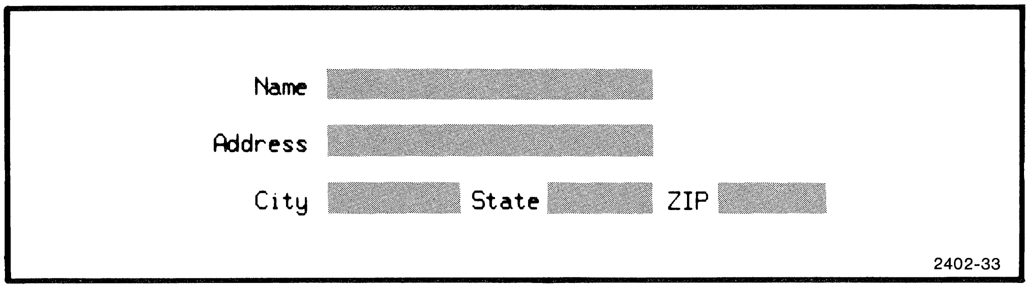
The terminal remembers its field separator when powered off or RESET.

Examples

!FIE @<CR> !FIE 64<CR>	Sets the field separator to the @ character, whose ADE is 64.
!FIE 9<CR>	Sets the field separator to the ASCII character 9.
!FIE 09<CR>	Sets the field separator to the ASCII HT (horizontal tab) character, whose ADE is 09.
!FIE<CR>	Sets the field separator to NUL. When data in a field is sent to the computer, no field separator is used.

Some Sample Transmissions

Suppose the following form begins in row 1 of the workspace. The unprotected fields are enhanced (shown here shaded gray); the last unprotected field has logical attribute numeric. The end of each non-blank line is at the end of the last unprotected field in the line. The three lines containing unprotected fields are separated from each other by blank lines.



Name ▨

Address ▨

City ▨ State ▨ ZIP ▨

2402-33

To store this form in the computer, give the command

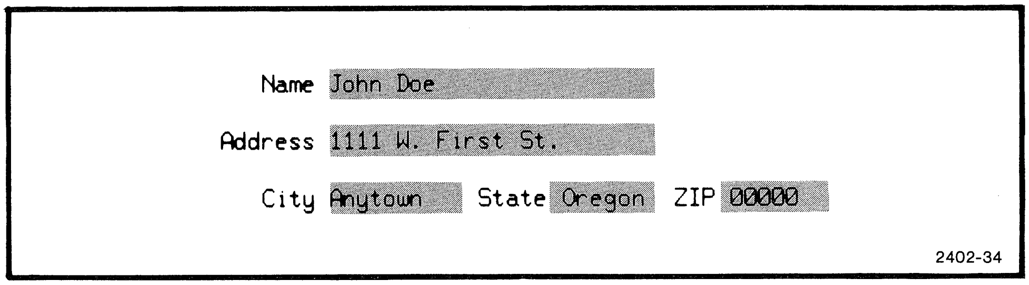
```
!SEN<CR>
```

The following information is sent to the computer:

```
!ATT P;––––Name !ATT EA;––––––––––––––––––––<CR> !ATT
P<CR>
!ATT P;–Address !ATT EA;––––––––––––––––––––<CR> !ATT
P<CR>
!ATT P:––––City !ATT EA;––––––––––!ATT SP;–State:!ATT;
––––––––––
!ATT SP;–ZIP !ATT EN;–––––<CR>
```

Each transmitted space is shown here as a dash. Remember that the default logical attribute of lines 2 and 4 (the blank lines) is alphanumeric. These lines must be protected to prevent text from being entered in them.

Suppose now the terminal is placed in form fillout mode and the form is filled out.



Name John Doe

Address 1111 W. First St.

City Anytown State Oregon ZIP 00000

2402-34

1. If no field separator is specified, the command

   ```
   !SEN A<CR>
   ```

   sends the following data to the computer:

   ```
   John Doe————————————<CR>1111—W.—First—St.————
   <CR>Anytown———Oregon——00000<CR>
   ```

   No field separator is used and each field is sent, including all trailing spaces. In a programming language which can divide an incoming line into blocks of predetermined length (such as COBOL), this is a convenient format.

2. Suppose the field separator is the number sign, #. The command

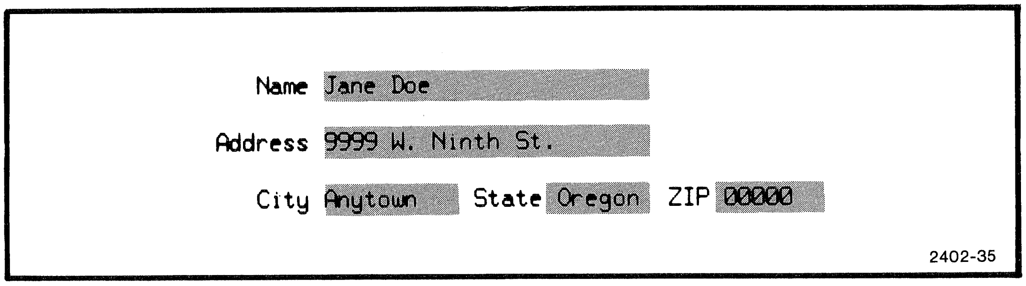
   ```
   !SEN A<CR>
   ```

   now sends the following to the computer

   ```
   #John—Doe<CR>#1111—W.—First—St.<CR>#Anytown#Oregon#00000<CR>
   ```

   The host program must use the # character to distinguish data from different fields.

3. Suppose that the same form is filled out for John Doe's sister, Jane Doe, who lives at a different street address in Anytown. Instead of erasing the form, the operator presses the HOME key to return the cursor to the first unprotected field, and simply types over the old information which must be changed.



Name Jane Doe

Address 9999 W. Ninth St.

City Anytown State Oregon ZIP 00000

2402-35

Now the first two unprotected fields are flagged with the logical attribute M. The SEND MOD command sends the data in these fields to the computer.

If no field separator is specified, the command

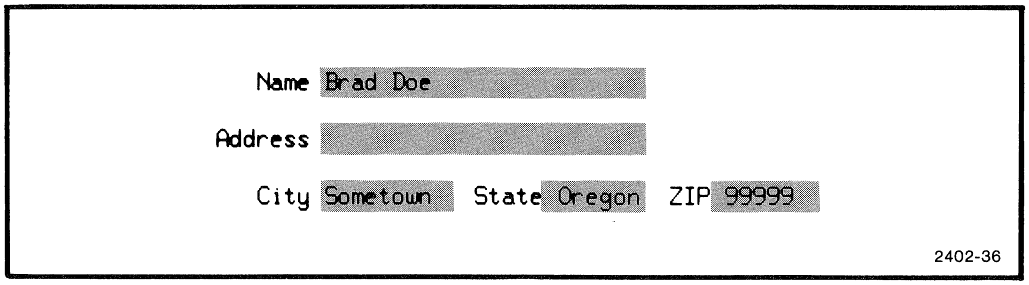
```
!SEN M<CR>
```

sends the following data to the computer:

001,010Jane—Doe——————————————<CR>003,0109999W.—Ninth—St.<CR>

Note that no spaces or other characters separate the row and column identifiers from the first character in the field.

4. Finally suppose the form is filled out for John's brother, Brad Doe, with no street address information provided, and the City and ZIP information modified:



Name Brad Doe

Address

City Sometown State Oregon ZIP 99999

If the field separator is the # character, the command

```
!SEN M<CR>
```

sends the following data to the computer:

#001,010#Brad—Doe<CR>#003,010<CR>#005,010Sometown#005,038 99999<CR>

Section 8

TEXT EDITING

THE TEXT-EDITING COMMANDS

The terminal recognizes four commands designed specifically for text editing: DCHAR (Delete Character), ICHAR (Insert Character), DLINE (Delete Line), and ILINE (Insert Line).

NOTE

If an editing command is typed on the keyboard and text from the keyboard is printed in the monitor, execution of the command inserts a blank line just below the line on which the command is typed. All examples deal with the workspace display.

DCHAR (Delete Character)

Syntax

```
    !DCHar [<count>]<CR>
```

where <count> is a positive integer. If <count> is not specified, it defaults to one.

Action

This command deletes <count> characters, beginning with the character at the cursor position. As each character is deleted, characters to the right of the cursor shift left to fill the gap. The cursor does not move. If the terminal is in form fillout mode, only characters to the right of the cursor in the same field shift left. If the terminal is not in form fillout mode, all characters right of the cursor on the same line shift left.

This command is equivalent to pressing the DELETE CHARACTER key <count> times.

Examples

Suppose the following text is displayed in the workspace, with the cursor position as indicated:

```
Everything seems seems in order.
                 ■
```

The command

```
!DCH<CR>
```

or

```
!DCH1<CR>
```

deletes the s at the cursor position, leaving the following display:

```
Everything seems eems in order.
                 ■
```

The subsequent command

```
!DCH5<CR>
```

leaves the desired display:

```
Everything seems in order.
                ■
```

Suppose a form contains incorrect information in an unprotected field; with the cursor positioned as shown:

```
Name:   Jane Doe DoeAge:   23
                ■
```

The command:

```
!DCH4<CR>
```

deletes the middle "Doe" and the extra space. Neighboring fields are not affected:

```
Name:   Jane Doe    Age:   23
                ■
```

ICHAR (Insert Character)

Syntax

```
!ICHar<CR>
```

Action

The ICHAR command places the terminal in insert mode. This command is equivalent to pressing the INSERT MODE key.

In insert mode, when new text is sent from the computer or typed on the keyboard, the cursor, the character at the cursor position, and characters to the right of the cursor are shifted right to make room for the new text.

Suppose the text

```
END PAGE
    ■
```

is displayed in the workspace, with the cursor positioned as shown. If the string

```
!ICH;OF<CR>
```

is sent from the computer, it inserts the text OF and displays the text

```
END OF PAGE
       ■
```

in the workspace, with the cursor positioned as shown.

If the string

```
!ICH;OF<CR>
```

is typed from the keyboard, the <CR> is sent to the workspace as text, and the cursor is positioned at the beginning of the next line:

```
END OF PAGE
■
```

In form fillout mode, only characters in the unprotected field containing the cursor are shifted right. Characters shifted past the rightmost position in that field are lost.

The DCHAR key or the !DCH command from the host can be used to delete unwanted characters at or to the right of the cursor position, *WITHOUT* leaving insert mode.

Suppose the following text is displayed in the workspace, with the cursor positioned as shown:

```
Everything would seems to be in order.
           ■
```

The command (from the host only)

```
!DCHAR 6<CR>
```

results in the following display:

```
Everything seems to be in order.
           ■
```

The terminal does not leave insert mode. Pressing the DELETE CHARACTER key six times gives the same results.

Any other cursor movement, resulting either from giving a command or from pressing a key, will cause the terminal to leave insert mode.

Suppose the terminal is in form fillout mode and the following form is displayed, with the cursor positioned as shown. The only unprotected fields are the three underlined fields; all other fields are protected.

```
NAME:  Ebenezer Scrooge  Age:  77
               ■
Position Applied for:  Miser
```

If the string

```
!ICH; A<CR>
```

is sent from the computer the following display results

```
NAME:  Ebenezer A ScroogeAge:  77
                  ■
Position Applied for:  Miser
```

The subsequent string

```
!ICH;ber<CR>
```

sent from the computer results in the form fillout display

```
Name:  Ebenezer Aber ScroAge:  77
Position Applied for:  Miser
```

Finally, the string:

```
!ICH;nathy
```

sent from the computer, moves the cursor past the end of the first unprotected field and into the second unprotected field. The following first lines of the form will be seen in rapid succession:

```
Name:  Ebenezer Abern ScrAge:  77
Name:  Ebenezer Aberna ScAge:  77
Name:  Ebenezer Abernat SAge:  77
Name:  Ebenezer Abernath Age:  77
Name:  Ebenezer AbernathyAge:  77
```

If the second unprotected field has the A (alphanumeric) attribute, any further insertion of characters shifts characters in the second field to the right and the old characters are lost. If the string

```
!ICH;B<CR>
```

is sent from the computer, the following display results:

```
Name:  Ebenezer AbernathyAge:  B7
Position Applied for:  Miser
```

However, if the second unprotected field has the N (numeric) attribute, a subsequent ICHAR command which inserts alphabetic characters will move the cursor to the first position in next unprotected field of the form and insert characters there.

If the string

```
!ICH;B.<CR>
```

is sent from the computer, the following display results:

```
Name:  Ebenezer AbernathyAge:  77
Position Applied for:  B. Miser
                         ■
```

If the insert character operation moves the cursor past the last unprotected field on the form, the cursor moves to the beginning of the first unprotected field which can accept the new characters; the new characters are inserted in that field.

DLINE (Delete Line)

Syntax

```
!DLIne [<count>]<CR>
```

where <count> is a positive integer. If <count> is not specified, it defaults to one.

Action

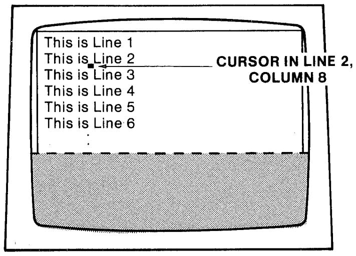
If the terminal is not in form fillout mode, this command deletes <count> consecutive lines of text, including the line containing the cursor. If the cursor is in the middle of a line, the entire line is deleted. As each line is deleted, the lines below roll up to fill the gap. The cursor moves to the beginning of the line which rolls up.

In form fillout mode, this command erases the contents of all unprotected fields in <count> lines of the form. The line containing the cursor is counted as a deleted line, whether or not it contains any unprotected fields. After this line, only lines containing at least one unprotected field are counted as deleted lines. The cursor is positioned at the beginning of the next unprotected field, after the last field erased.

This command is equivalent to pressing the DELETE LINE key six times.

Examples

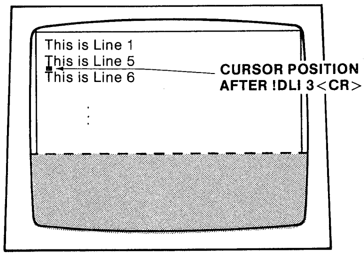
Suppose the terminal is not in form fillout mode, and the workspace contains the text shown, with the cursor in line 2, column 8:



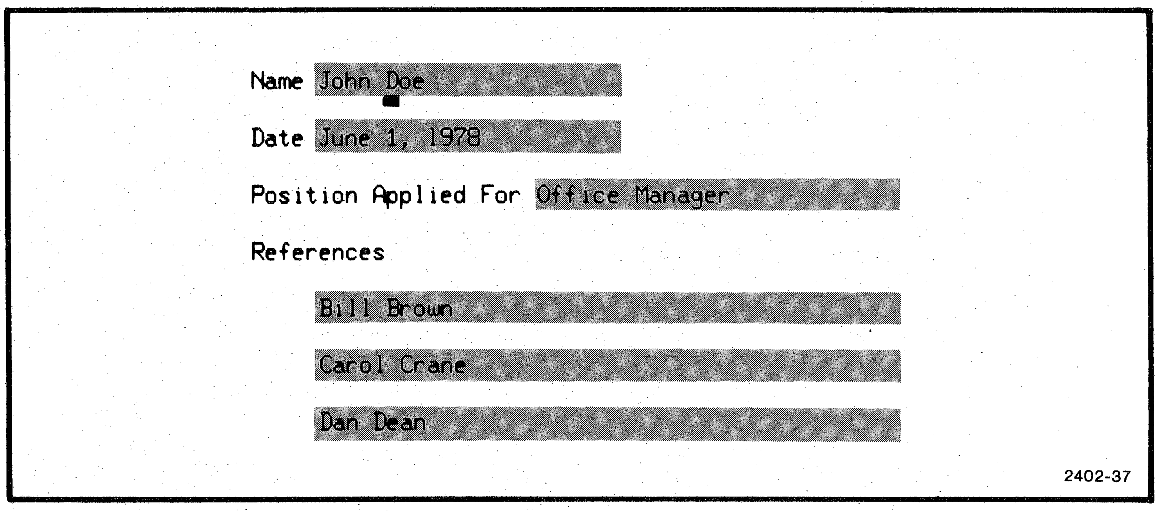
The command

```
!DLI 3<CR>
```

gives the workspace display shown here:



Suppose the terminal is in form fillout mode, and the workspace holds the form shown here. The unprotected fields are enhanced (shown here shaded gray):



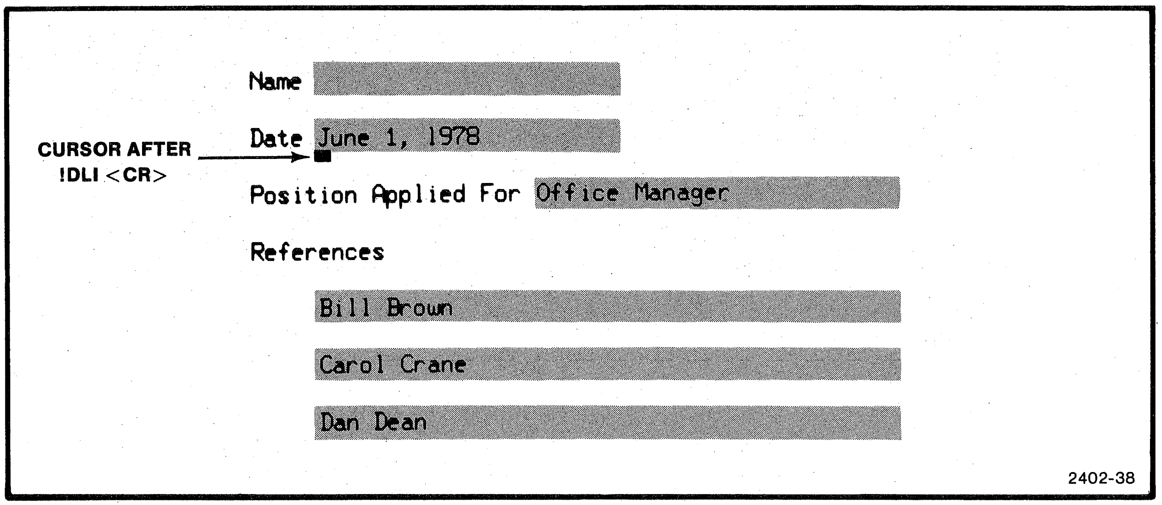
```
Name John Doe

Date June 1, 1978

Position Applied For Office Manager

References

          Bill Brown

          Carol Crane

          Dan Dean
```

2402-37

If the cursor is positioned anywhere in the first line of the form, the command

!DLI< CR>

results in the display show below.



```
Name

Date June 1, 1978

Position Applied For Office Manager

References

          Bill Brown

          Carol Crane

          Dan Dean
```

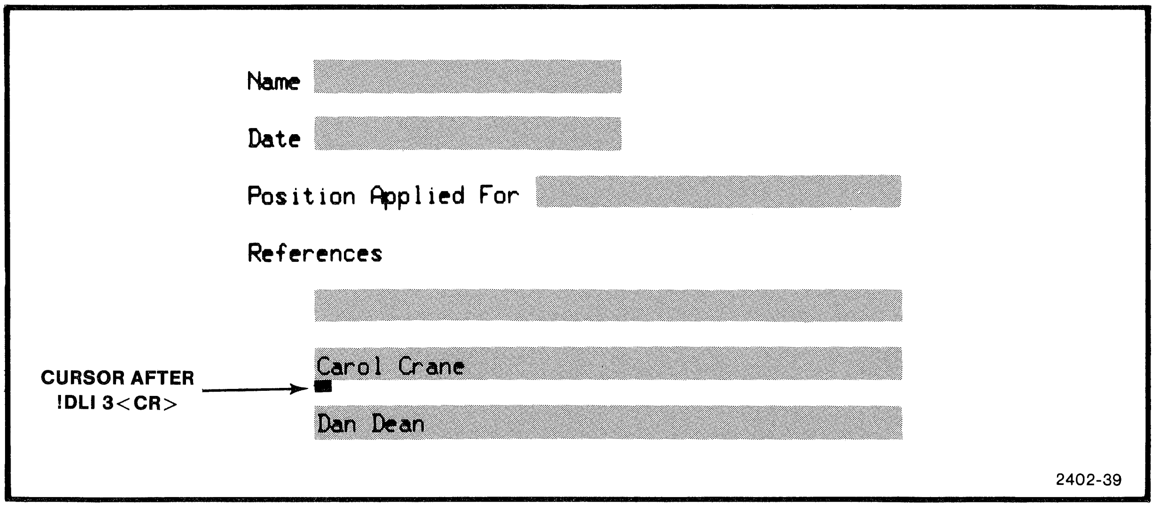
CURSOR AFTER
!DLI <CR>

2402-38

The subsequent command

```
!DLI 3<CR>
```

results in the display shown here. Note that the line "References:" is not counted as a deleted line, since it contains no unprotected fields.



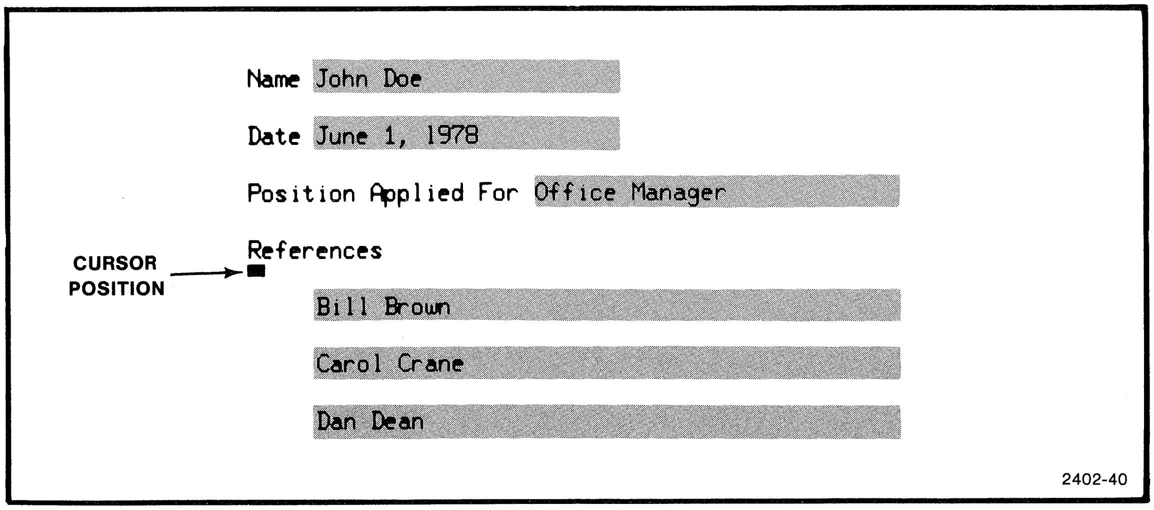
Name
Date
Position Applied For
References

Carol Crane
Dan Dean

CURSOR AFTER !DLI 3<CR>

2402-39

Suppose you begin with the form below and the cursor positioned as shown.



Name John Doe
Date June 1, 1978
Position Applied For Office Manager
References
Bill Brown
Carol Crane
Dan Dean

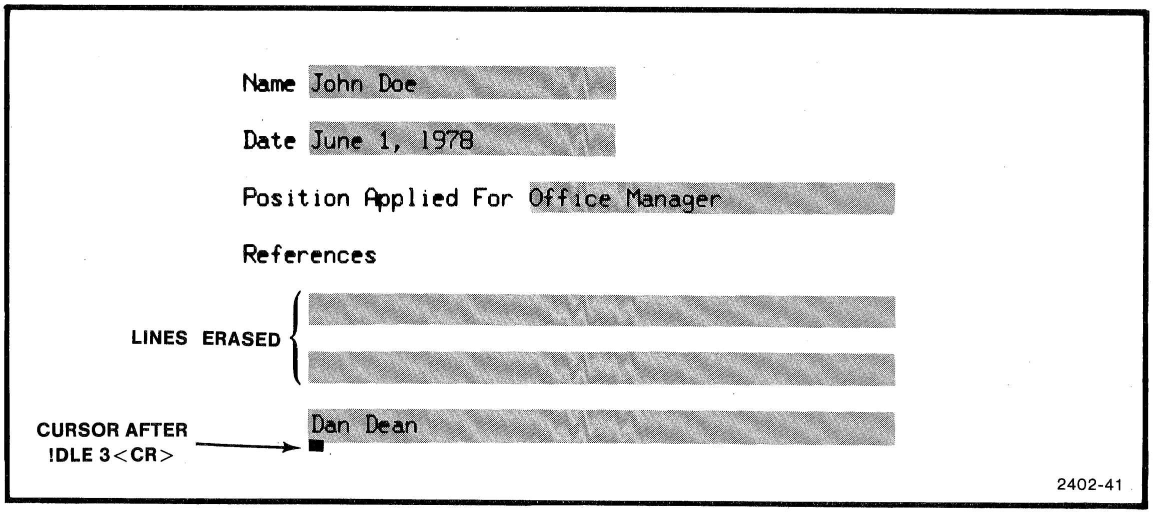
CURSOR POSITION

2402-40

The command

```
!DLI 3<CR>
```

results in the display shown here. Observe that the line which originally contained the cursor has been counted as the first deleted line, even though it contains no unprotected fields.



Name John Doe

Date June 1, 1978

Position Applied For Office Manager

References

Dan Dean

LINES ERASED

CURSOR AFTER !DLE 3<CR>

2402-41

ILINE (Insert Line)

Syntax

```
!ILIne [<count>]<CR>
```

where <count> is a positive integer. If <count> is not specified, it defaults to one.

Action

This command inserts <count> blank lines into the text immediately below the line containing the cursor. The cursor is positioned at the beginning of the newest line. Lines of text below the cursor position are rolled down to make room for the inserted blank lines, and the scroll is lengthened so that these lines are saved on the display list.

This command is equivalent to pressing the INSERT LINE key <count> times.

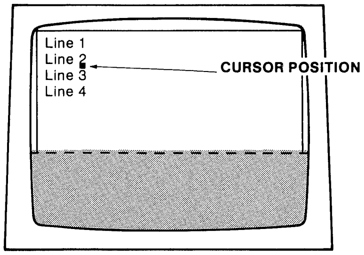
The ILINE command makes it easy to insert new text between lines of old text. Use the ILINE command to create several blank lines at the desired location. Type the new information into the blank lines and use the DLINE command to delete any blank lines left over.

NOTE

For text editing applications, the first line entered into the workspace should be blank. If new text must be inserted above the old text, the cursor is moved to the beginning of the workspace and the ILINE command is used to create space for the new text. If the first line of the workspace already contains text, this procedure inserts blank lines below the first line of old text, rather than above it.

Examples

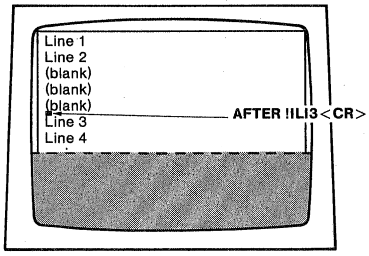
Suppose the workspace contains the text shown opposite, with the cursor positioned as shown:



The command

```
!ILI 3<CR>
```

inserts three blank lines between line 2 and line 3, leaving the cursor in column 1 of the newest blank line:



When the terminal is in form fillout mode, the ILINE command has no effect.

Section 9

GRAPHICS

The terminal with Option 25 or 26 has basic graphic capability. A terminal thus equipped can draw several styles of vectors (line segments), intermix graphics with text and forms, and store special purpose character fonts defined by the user. This section explores these capabilities.

THE GRAPHICS COMMANDS

There are seven commands designed for creating graphic displays. These are the GRAPHIC, VECTOR, RVECTOR, LINE, STRING, ERASE G, and SHRINK commands. Each of these is discussed in turn.

GRAPHIC

(Requires Option 25 or 26)

Graphics are displayed in the workspace. Before this can be done, the workspace must be prepared to display graphs by defining a graphic region. The GRAPHIC command is used for this purpose.

Syntax

```
    !GRAphic <beg row> <end row> [<beg col> [<end col>]] <CR>
```

where all parameters are positive integers designating rows and columns *in absolute workspace coordinates*. Thus <beg row> must be less than <end row>, <beg col> must be less than <end col>, and <end col> must be less than or equal to 80. Also, <end row> must **not** exceed <beg row> by more than 52, since the graphics region cannot exceed 53 rows in length. The default values of <beg col> and <end col> are 1 and 80, respectively.

Action

This command defines a graphic region in the workspace and erases all information currently stored in this region. The graphic region thus defined consists of rows <beg row> through <end row>, and columns <beg col> through <end col> in each of these rows.

Examples

!GRA 1,35<CR>	Creates a graphic region in the workspace containing columns 1 through 80 of rows 1 through 35.
!GRA 1,35,30<CR>	Creates a graphic region in the workspace containing columns 30 through 80 of rows 1 through 35.

The structure of a graphic region is best illustrated by an example. The command

```
!GRA 10,19,20,49<CR>
```

creates a graphic region which occupies rows 10 through 19, columns 20 through 49 in each of these rows.

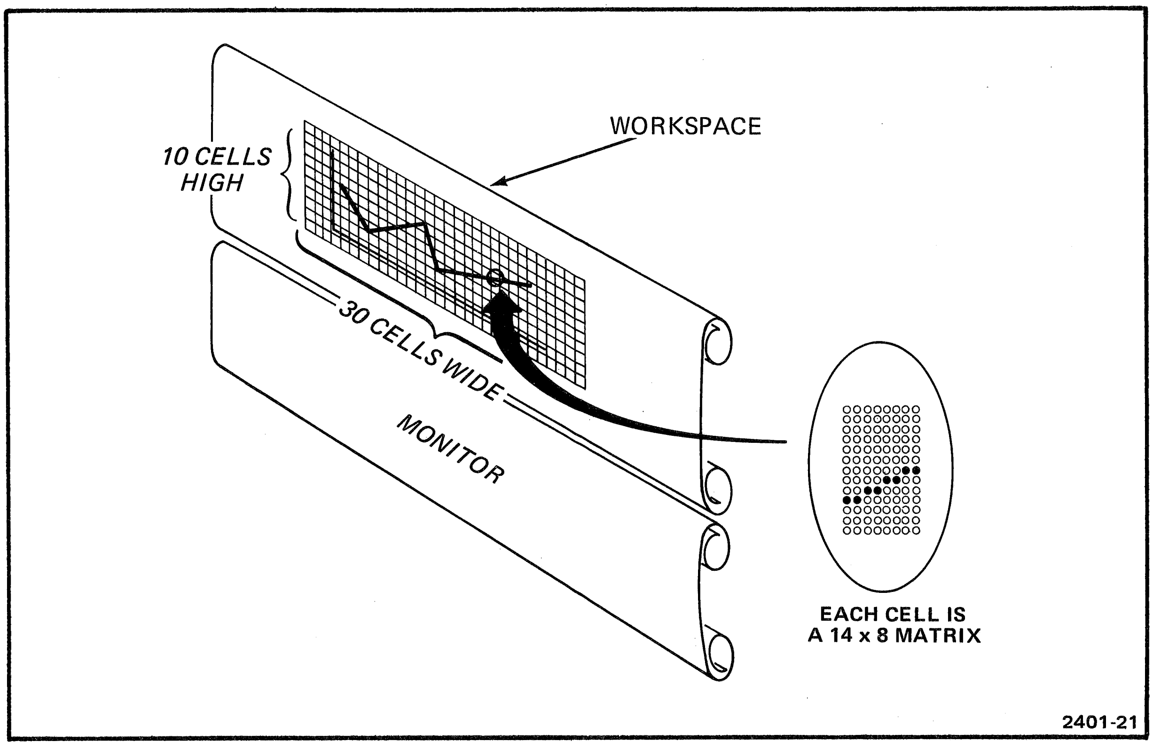


Figure 9-1. A Graphic Region.

As illustrated in Figure 9-1, this graphic area is 10 cells (character cells) high and 30 cells wide. Each cell consists of a *dot matrix* 8 dots wide by 14 dots high. Each dot can be turned on (lighted) by an electronic *vector beam*. Various commands discussed in this section draw vectors or display user-defined symbols by turning on patterns of these dots.

The columns of dots are numbered from left to right across the graphic region, starting with 0 for the leftmost column, and from bottom to top, starting with 0 for the bottom row. In Figure 9-1, the 240 columns of dots (30 cells, each cell 8 dots wide) are numbered from 0 to 239; the 140 rows of dots (10 cells, each cell 14 dots high) are numbered from 0 to 139. This establishes a coordinate system in the graphic region. Each dot in this region is assigned X and Y-coordinates. The X-coordinate gives the dot's horizontal position; the Y-coordinate gives the dot's vertical position. These coordinates are used in the VECTOR and RVECTOR commands.

It is possible to define more than one graphics region in the workspace. If this is done, new graphic commands affect only the graphic region most recently defined. Different graphic regions should not overlap.

ENABLE

(Requires Option 25 or 26 and Option 38)

The ENABLE command places the terminal in the Graphic Input (GIN) mode. This mode is used to provide beam position information to the host computer.

Syntax

```
!ENAble [<count>]<CR>
```

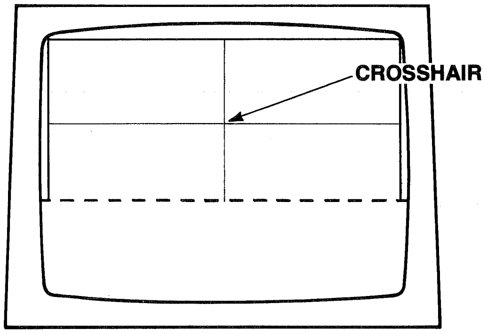
where <count> is a positive integer specifying the number of points to be sent to the host computer. If <count> is not specified, it defaults to infinity.

NOTE

GIN mode may also be initiated by pressing the crosshair key.

Action

The ENABLE command causes the terminal to enter GIN mode. When GIN is first ENABLED, the crosshair is displayed at the graphic beam position. The crosshair can then be manipulated with the cursor control and home keys.



A crosshair appears in the workspace when the 4025A is ENABLED.

When a key or other than the crosshair control key is pressed, a report is sent to the host. The report is in the form:

<cmd. chr.> DAT 03, <key>, <x pos>, <y pos>;

where

<cmd. chr.> is the current command character. DAT 03 indicates the crosshair device.

<key> is the ASCII decimal equivalent of the key value that generated this report.

<x pos> is a three-digit number indicating the location of the crosshair with respect to the horizontal axis.

<y pos> is a three-digit number indicating the location of the crosshair with respect to the vertical axis.

The terminal remains in GIN mode until one of the following occurs:

- the crosshair key is pressed.
- a DISABLE command is sent from the host or typed on the keyboard.
- the specified <count> number of points and the carriage return have been sent. An end-of-line sequence is not sent between each point when multiple points are sent.

Certain characteristics of the graphic beam during GIN mode should be noted. When ENABLE is given, the crosshair is displayed at the graphic beam position. If the crosshair is moved, the graphic beam is moved to the crosshair position when a key is pressed. Also, if INKing is on, a line is drawn from the previous graphic beam position to the present crosshair position when a key is pressed. The key normally used to set the graphic beam at the position of the crosshair is the pad terminator key. However, pressing most of the alpha or numeric keys will have the same result. Keys which, when pressed, do not set the graphic beam at the position of the crosshair are the BREAK, CROSSHAIR, SHIFT, CONTROL, HOME, TTY LOCK, NUMERIC LOCK, and COMMAND LOCK OUT keys.

Examples

!ENAble<CR>	Places the terminal in GIN mode and sets the crosshair at the graphic beam position.

art here

!ENA 5<CR>	Places the terminal in GIN mode for the specified number of points (5) and sets the crosshair at the graphic beam position. An end-of-line sequence is sent, after five reports have been sent, which causes the crosshair to leave the screen and the terminal to leave GIN mode.

DISABLE

(Requires Option 25 or 26 and Option 38)

Syntax

```
!DISable<CR>
```

Action

The DISABLE command removes the terminal from GIN mode. The crosshair is removed from the graphic area and the crosshair control keys return to controlling the alpha cursor. An end-of-line sequence is sent to the host as the terminator of the GIN messages, if any have been sent.

VECTOR

(Requires Option 25 or 26)

When a graphic area of suitable size has been defined, vectors (line segments) can be drawn in the graphic area using the VECTOR command.

Syntax

```
!VECtor <X0><Y0><X1><Y1>[<X2><Y2> ...<Xn><Yn>]<CR>
```

where all $<X_i>$ and $<Y_i>$ parameters are positive integers. The parameters are separated by spaces or commas.

Action

This command draws a vector from the point with graphic coordinates ($<X_0>$, $<Y_0>$) to the point with coordinates ($<X_1>$, $<Y_1>$). If additional pairs of coordinates are specified, additional vectors are drawn from ($<X_1>$, $<Y_1>$) to ($<X_2>$, $<Y_2>$), from ($<X_2>$, $<Y_2>$) to ($<X_3>$, $<Y_3>$), and so on.

The $<X_i>$ and $<Y_i>$ coordinates are graphic area coordinates and must lie within the coordinate system defined by the current graphic area. If the graphic area was defined by the !GRA 10,19,20,49<CR> command, all $<X_i>$ coordinates should be between 0 and 239, and all $<Y_i>$ coordinates between 0 and 139.

If a point is specified which lies outside the graphics area, the terminal does not try to draw a line to that point. It resumes drawing with the next line segment which lies entirely within the graphics region.

Example

Suppose you have used the !GRA 10,19,20,49<CR> command to define the 240 X 140 graphic area described earlier. The command

```
!VEC 120,120 91,30 168,85 72,85 149,30 120,120<CR>
```

creates the display shown in Figure 9-2. (Axes are *not* shown on the display.) Note that, since either a space or a comma serves as the separator, we have alternated these to emphasize the VECTOR coordinate pairs.

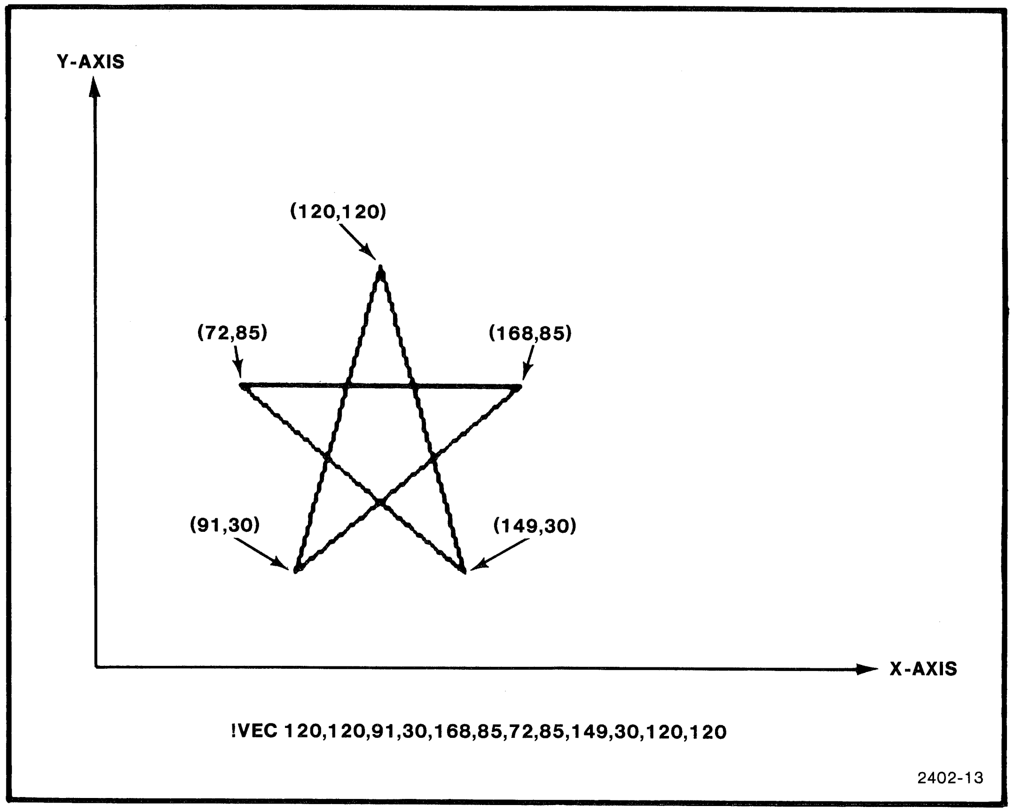


Figure 9-2. The VECTOR Command.

RVECTOR (Relative Vector)

(Requires Option 25 or 26)

It is possible to draw vectors by specifying *relative coordinates* — that is, coordinates relative to the last vector beam position. This is done by using the RVECTOR command.

Syntax

!RVEctor <rel X_0><rel Y_0><rel X_1><rel Y_1> [<rel X_2><rel Y_2> ...
<rel X_n><rel Y_n>]<CR>

where <rel X_i> and <rel Y_i> are integers, not necessarily all positive. The parameters are separated by spaces or commas.

Action

This command draws one or more vectors in the graphics area, as does the VECTOR command. The pair <rel X_0>, <rel Y_0> specifies coordinates relative to the current vector beam position. Each succeeding pair of <rel X_i>, <rel Y_i> parameters specifies new coordinates *relative to the preceding coordinate pair.*

Example

Suppose that the current vector beam position is at the point with absolute workspace coordinates (120,65). The command:

```
!RVE 0,55 −29,−90 77,55 −96,0 77,−55 −29,90<CR>
```

draws the star in Figure 9-3. It is the same figure drawn by the earlier VECTOR command, but now each pair of coordinates given is relative to the preceding pair of coordinates.

As in the VECTOR command, if a pair of coordinates specifies a point outside the graphic area, the terminal ignores that point and resumes drawing with the next vector lying entirely within the graphic area.

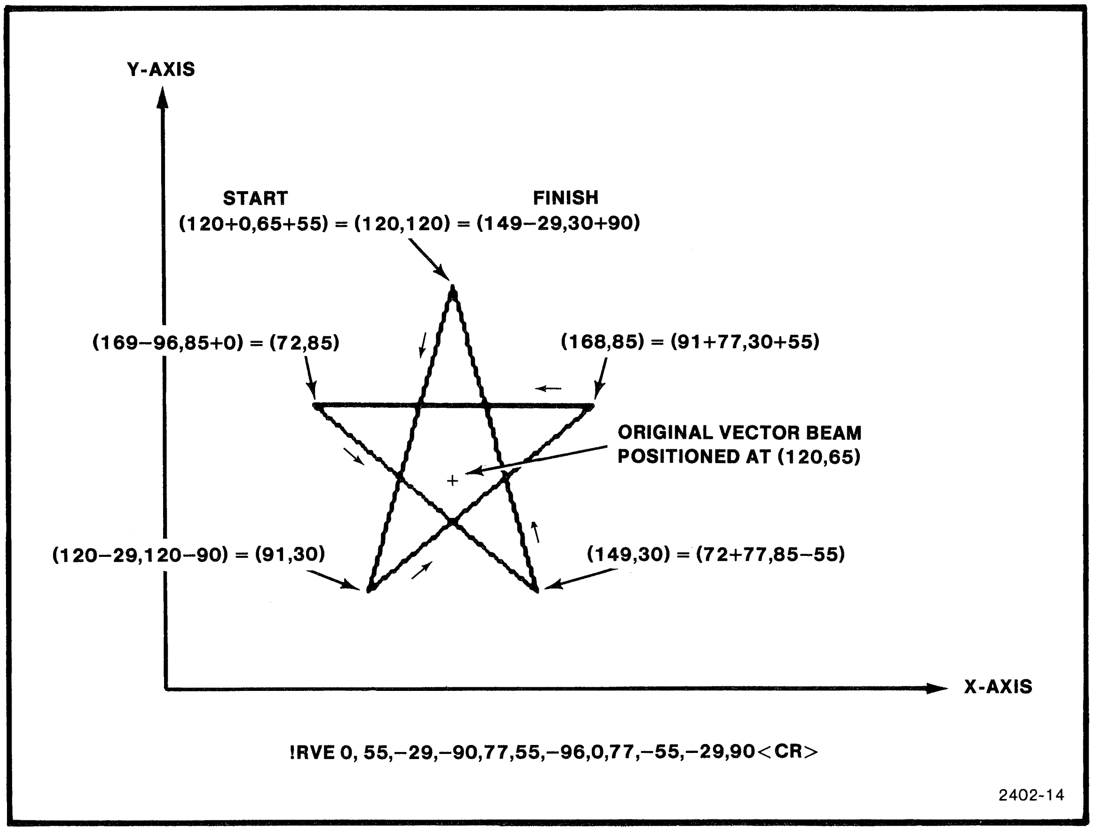


Figure 9-3. The RVECTOR Command.

LINE

(Requires Option 25 or 26)

The terminal can draw different styles of vectors. The style of vector to be drawn is selected with the LINE command.

Syntax

```
!LINe [<line type>]<CR>
```

where <line type> must be one of the following:

- A digit from 1 to 8, inclusive
- The letter P
- The letter E

If <line type> is not specified, it defaults to one.

Action

This command sets the type of line used to draw vectors in subsequent VECTOR and RVECTOR commands. Line type 1 is a solid line, the default line type. Line types 2 through 8 are various styles of dashed lines. Line types 1 through 8 are shown in Figure 9-4.

Line type P causes subsequent VECTOR and RVECTOR commands to plot isolated points rather than connect points with line segments.

Line type E causes subsequent VECTOR and RVECTOR commands to draw "dark vectors" which erase any lighted points through which they pass.

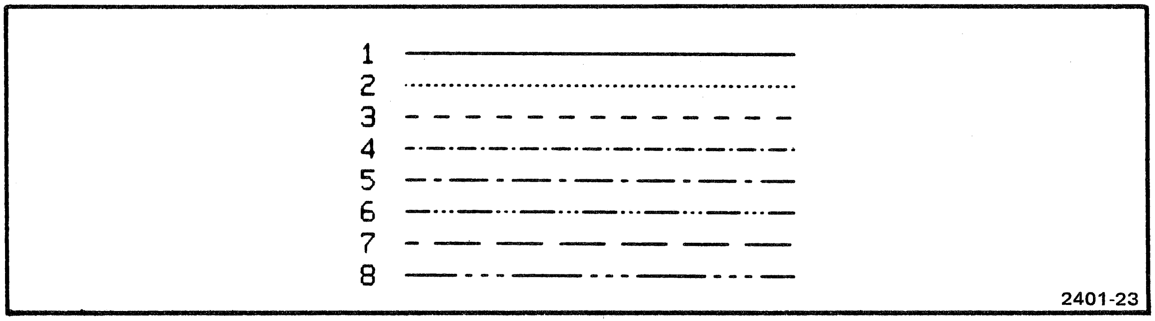


Figure 9-4. Vector LINE Types.

CIRCLE

(Requires Option 25 or 26 and Option 38)

The CIRCLE command is used to create circles, circle sections, and equilateral polygons.

Syntax

```
!CIRcle <radius> [<start angle>][<end angle>][increment angle]<CR>
```

where

<Radius> is a positive integer representing the radius of the circle or polygon in raster units.

<Start angle> is a positive or negative integer which states the angle at which the first radius of the circle will be drawn.

<End angle> is a positive or negative integer which states the angle at which the last radius of the circle will be drawn.

<Increment angle> is a positive integer which represents the number of degrees between the vertices of the polygon.

Action

The CIRCLE command causes a shape to be drawn around the beam position. The circle (or polygon) will be drawn from the <start angle> to the <end angle> at a radius of <radius> raster units. If the <start angle> and <end angle> are not given, they default to 0 and 360 degrees, respectively.

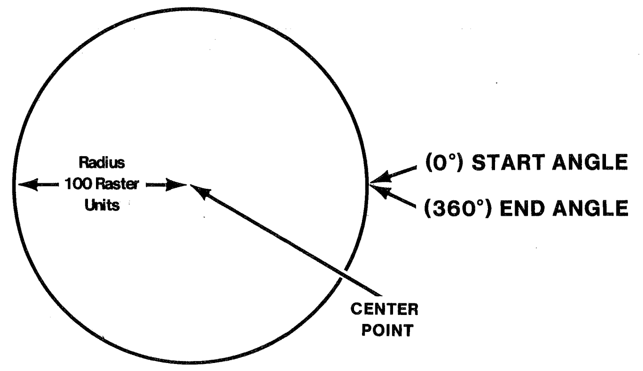
If <increment angle> is given, the CIRCLE command will mark vertices at intervals of <increment angle> degrees. The vertices then are joined to form a polygon. Default value for <increment angle> is 4 degrees.

All angles are measured with 0 degrees as the point of reference. Zero degrees is the horizontal line segment which extends from the center point to the right side of the graphic area. Angle values increase in a direction moving counter clockwise relative to zero degrees.

Examples

!CIRcle 100 0 360<CR>
!CIR 100<CR>

Creates a complete circle with a <radius> of 100 raster units, as shown below.

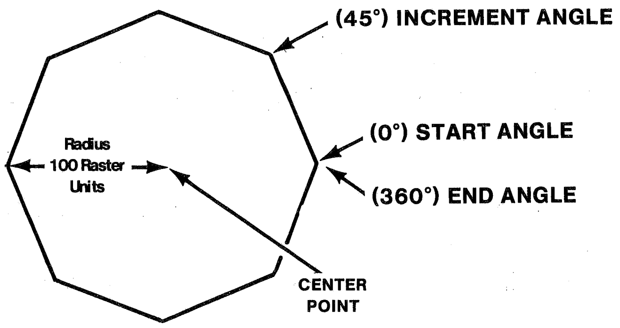


CIRCLE CREATED BY !CIR 100 0 360<CR>

2657-17

!CIR 100 0 360 45<CR>

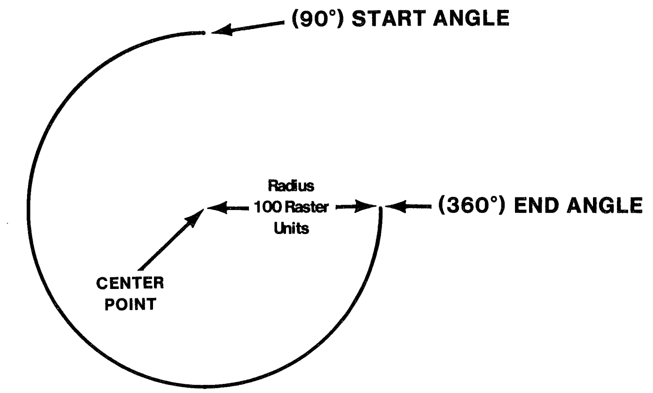
Creates a polygon with a <radius> of 100 raster units. As shown below, including the <increment> of 45 degrees causes an eight-sided polygon to be formed. >Start angle> and <end angle> must be given when <increment angle> is used.



POLYGON CREATED BY !CIR 100 0 360 45<CR>

2657-18

!CIR 100 90 360<CR>	Creates a sector of a circle with a <radius> of 100 raster units from the <start angle> of 90 degrees to the <end angle> of 360 degrees, as shown below.



SHAPE CREATED BY !CIR 100 90 360 <CR>

2657-19

INK

(Requires Option 25 or 26 and Option 38)

The INK command enables the drawing of lines between points in the graphic area without typing in coordinates. The terminal must be in the GIN mode to INK.

Syntax

```
!INK [Yes | No]< CR>
```

If no parameter is specified, Yes is assumed.

Action

When the INK or INK YES command is given, the terminal can draw lines from the present crosshair location to the previous location without designating the coordinates as in a VECTOR or RVECTOR command. The terminal must be ENABLED by giving the ENABLE command or pressing the zero/crosshair key. After this has been done, pressing the pad terminator key or any other non-cursor moving key causes a line to be drawn from the present position of the crosshair to the previous position.

The INK NO command turns INKing off.

Example

```
!INK YES< CR>
```

Refer to Figure 9-5. When drawing a line from crosshair position one to position two, position one must first be established by moving the crosshair to the desired location and pressing the pad terminator key. Remember that the crosshair is displayed by giving the ENABLE command or pressing the crosshair key. When the crosshair first comes up, if INKing is already on, a line is drawn from the previous beam position to the crosshair position when the pad terminator key is pressed.

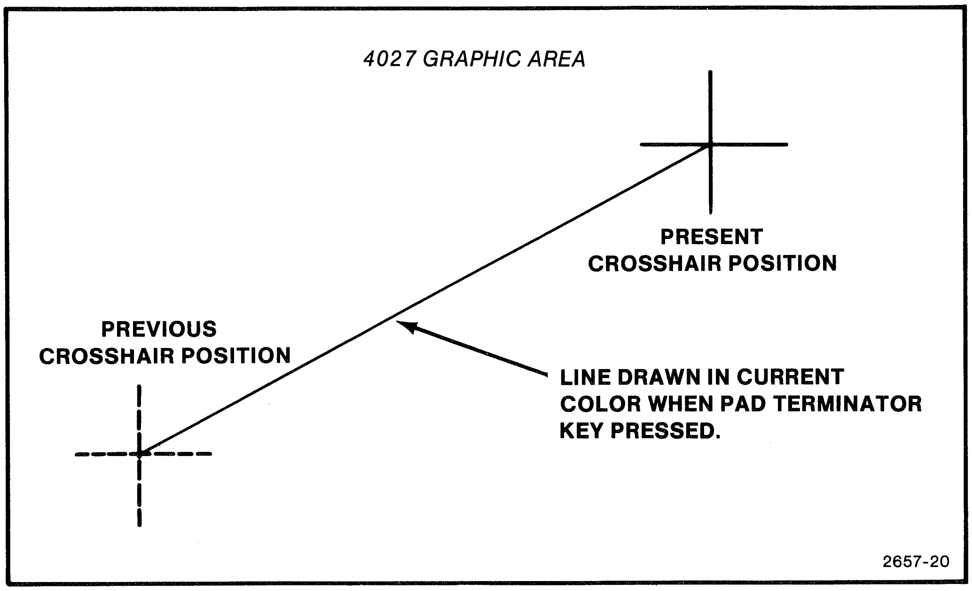

4027 GRAPHIC AREA
PRESENT CROSSHAIR POSITION
PREVIOUS CROSSHAIR POSITION
LINE DRAWN IN CURRENT COLOR WHEN PAD TERMINATOR KEY PRESSED.
2657-20

Figure 9-5. Drawing a Line in INK Mode.

After the position of the crosshair has been established, give the INK command. Then, each time the crosshair is repositioned and the pad terminator key is pressed, a line is drawn between the present location of the crosshair and the previous one.

```
!INK No<CR>
```

The INKing process is terminated. If INKing is off, then no vectors are drawn when points are set in GIN mode.

If an ENABLE command for X points is given, after X points are entered, the crosshair goes down and INKing appears to terminate. However, INKing is still in effect, and additional vectors will be INKed if the zero/crosshair key is pressed, returning the crosshair to the graphic region. INKing is terminated only by giving the INK NO command.

STRING

(Requires Option 25 or 26) ,

At times it may be desirable to enter text into a graphic area. Since there is no workspace cursor in the graphic area, and since straight text sent to the terminal from the computer is entered at the workspace cursor location, another way is needed to put text into a graphic area. The STRING command does this.

Syntax

```
!STRing <text> <CR>
```

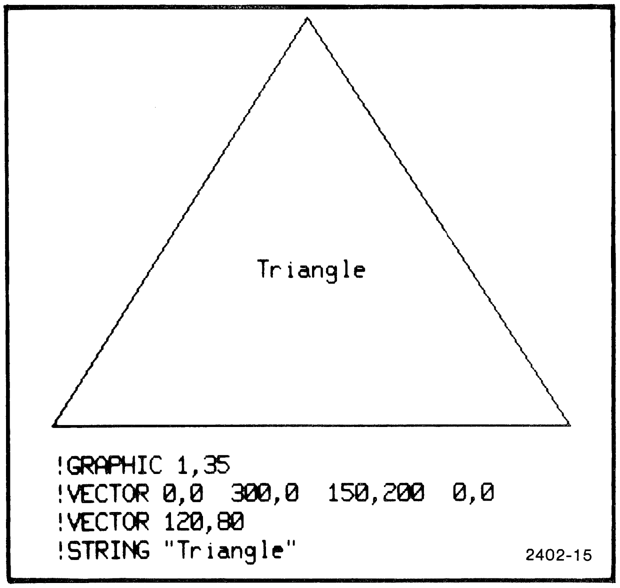
where <text> may be:

1. One or more delimited ASCII strings.
2. A sequence of ASCII Decimal Equivalents.
3. Any combination of 1 and 2.

The string defined by <text> should not contain the command character.

Action

This command inserts the string defined by the <text> parameter into the graphics area. The first character defined by <text> is displayed in the character cell containing the vector beam. Succeeding characters of <text> are displayed in succeeding character cells. Any vectors or characters that were previously displayed in the character cells where <text> is inserted are no longer visible, since each character of <text> fills an entire character cell. A sample STRING command and the resulting display is shown in Figure 9-6.



```
!GRAPHIC 1,35
!VECTOR 0,0  300,0  150,200  0,0
!VECTOR 120,80
!STRING "Triangle"
```

2402-15

Figure 9-6. The STRING Command.

ERASE G

(Requires Option 25 or 26)

When the information displayed in a graphics area is no longer needed, it can be deleted in one of two ways. One may delete the graphic region and all information stored in it from the workspace display list. One may also erase the graphic information but leave the graphic region defined to display new graphic information.

To delete the graphic region from the display list, give the ERASE WORKSPACE command. The graphic region, along with all other information in the workspace, is deleted from the display list. No further graphics commands can be executed until a new graphic region is defined by a GRAPHIC command.

If one wishes to reuse the same graphic region, the ERASE G command is used.

Syntax

```
!ERAse G<CR>
```

Action

This command erases the contents of the graphic region containing the vector beam. All information in the region, including text, is erased. The graphic region is still defined and new graphic information and text can be displayed there.

SHRINK

There are two circumstances which make it necessary for the terminal to alter the coordinates of graphic information in its display list.

- The terminal accepts 4010-style graphic commands from a host computer. In 4010-style graphic commands, the X-coordinates can be as great as 1023. The X-coordinates in a VECTOR or RVECTOR command should not exceed 639 (in a graphics region occupying all 80 columns). It is necessary, therefore, to scale incoming 4010-style graphic commands for display in the graphic region. (See discussion of 4010-style graphics in this section.)

- To make hard copies of a graph displayed on the screen, using the HCOPY command. (See the PERIPHERALS section.) In order for graphs to be properly proportioned on 4611 or 4631 Hard Copy Units not equipped with Option 31, the Y-coordinates must be scaled.

The SHRINK command is used to instruct the terminal to alter graphic coordinates for each of these circumstances.

Syntax

```
!SHRink [Yes|Hardcopy|Both|Resolution|No]<CR>
```

The default parameter is Yes.

Action

SHRINK YES. This command causes the terminal to "shrink" X- and Y-coordinates in subsequent VECTOR or RVECTOR commands, multiplying them by a factor of approximately 5/8. This accommodates the terminal to the range of possible coordinates in 4010-style graphics commands.

To use the terminal to execute a command file written for 4010-series terminals, first dimension the graphics area to hold 35 rows of 80 columns. (!GRA 1,35,1,80 or !GRA 10,44 are two GRAPHIC commands which do this.) Then give a SHRINK YES command to put the terminal in graphics shrink mode.

SHRINK HARDCOPY. (Required for 4611 04 4631 Hard Copy Units not equipped with Option 31). This command puts the terminal in hardcopy shrink mode. In this mode, the Y-coordinates (but not the X-coordinates) in VECTOR and RVECTOR commands are multiplied by a factor of 7/8. This pre-distorts graphs drawn on the screen, so they appear in the proper proportion when copied on a TEKTRONIX 4611 or 4631 Hard Copy Unit. Character strings inserted in the graphics area with the STRING command will not be shrunk. SHRINK HARDCOPY affects only vectors, not alphanumerics.

SHRINK BOTH. This command puts the terminal in both graphics shrink and hardcopy shrink modes. The X-coordinates in subsequent VECTOR and RVECTOR commands are multiplied by approximately 5/8, while the Y-coordinates are multiplied by approximately 35/64.

SHRINK NO. This command removes the terminal from both shrink modes.

SHRINK RESOLUTION. This command tells the terminal to translate VECTOR and RVECTOR commands, which may then be passed to the 4662/4663 Plotters with Option 04 in 4096 X 4096 resolution information.

EFFECTS OF A GRAPHIC REGION

The presence of a graphic region affects the action of some commands and keys, summarized here:

DELETE CHARACTER: Inside a graphic region, the character is replaced by a space.

DELETE LINE: In a line which passes through a graphic region, only characters outside the graphic region are deleted. Information inside the graphic region is not deleted.

ERASE & SKIP: In a line that passes through a graphic region, only characters outside the graphic region are deleted.

ERASE WORKSPACE: This erases the entire workspace, including the graphic region definition. A new GRAPHIC command must be given before further graphics can be displayed.

CURSOR MOVEMENT AND TYPING: The ASCII keys, the cursor movement keys and commands, and the scrolling keys and commands are not affected by the presence of the graphics region. If the cursor is moved into a graphic region and a character typed on the keyboard, that character replaces graphic information previously stored in that character cell.

FORM FILLOUT MODE: All locations within the graphic region are protected in form fillout mode. If a graphic region is less than 80 columns wide and no form exists in the side region(s), then the side region(s) are unprotected by default and text may be entered into them. To prevent text from being entered into these regions, they must be protected or the graphic region must be expanded to include all 80 columns.

ATTRIBUTE CODES: Inside a graphic region, the terminal inserts only font attribute codes in the display list. All other attributes are ignored. Any visual attributes (enhanced, etc.) which are in effect at the left edge of the graphic region affect the entire row of character cells running through the graphic region. Logical attributes and font codes in effect at the left edge of the graphic region do not affect the graphic region itself, but characters to the right of the graphic region are given these same font and logical attributes.

THE SEND COMMAND: Graphic information in a graphic region is not transmitted by the SEND command. Every character cell containing graphic information is transmitted as an ASCII space. (Text information **is** sent, however.) Suppose the graph shown in Figure 9-7 is displayed in the workspace.

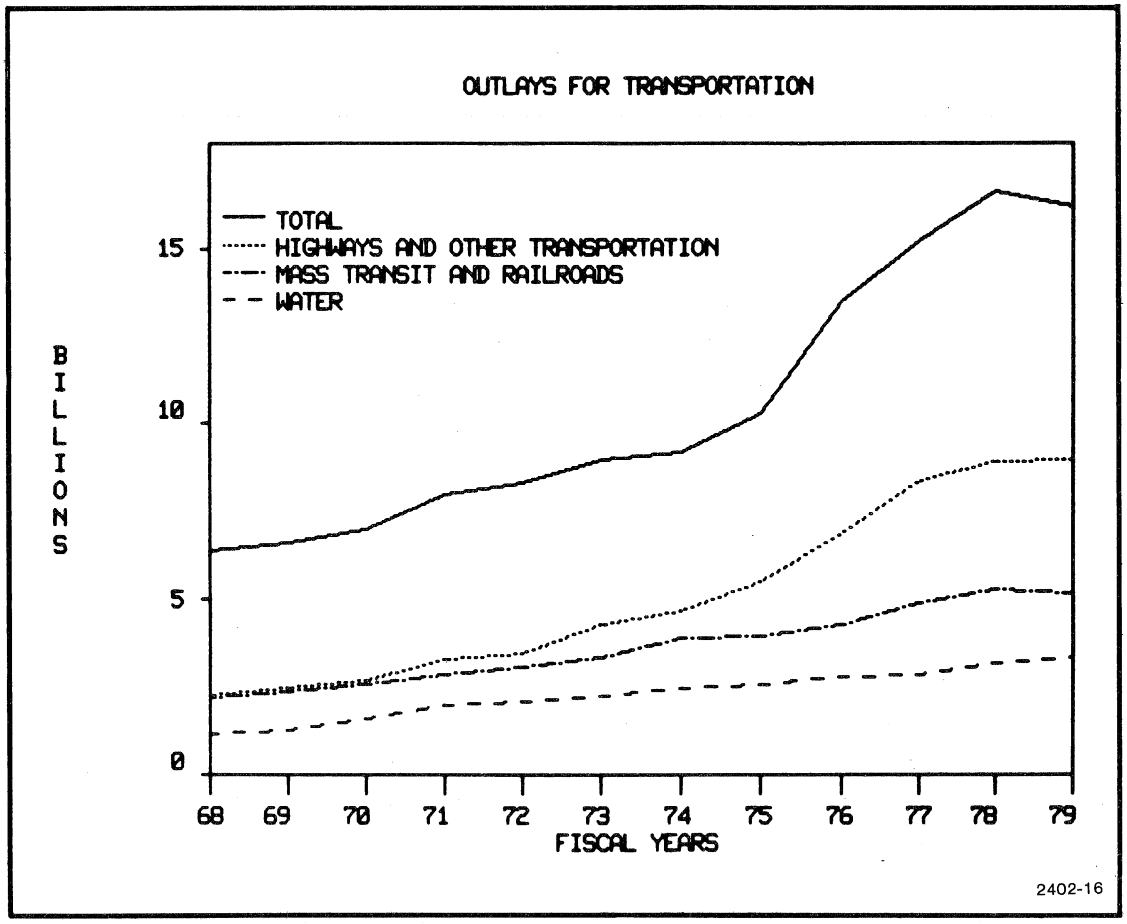


Figure 9-7. A Graphic Display.

If you do a SEND operation to the computer, then SEND back from the computer to the terminal, you obtain the display in Figure 9-8. No information generated by graphic commands was sent to the computer. The display in Figure 9-8 is what is stored in the computer.

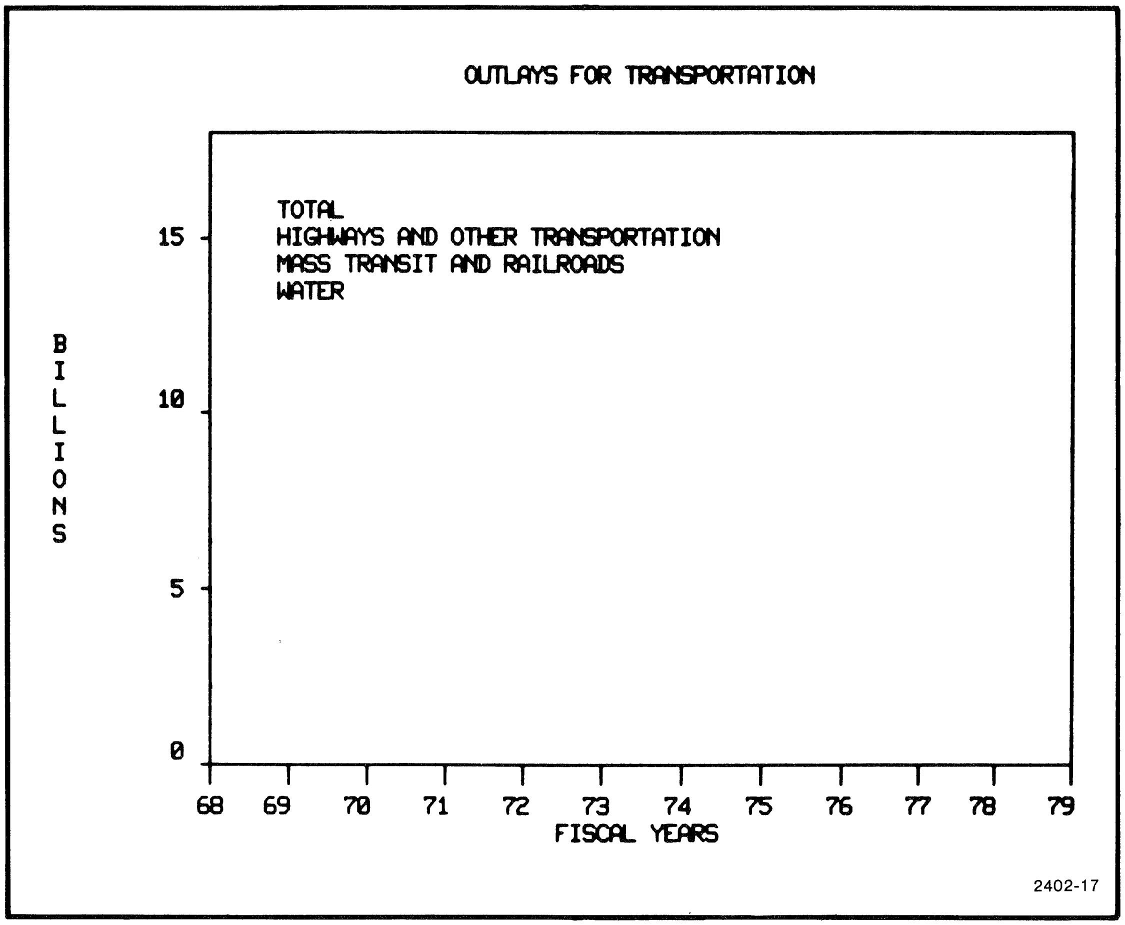


Figure 9-8. A Graphic Display After the SEND Command.

4010-STYLE GRAPHICS ON THE 4025A

The 4025A with a Graphics Memory option, accepts 4010-style graphic commands when these commands are sent from the host. (The terminal does not accept 4010-style graphic commands entered on the keyboard.) 4010-style graphics uses ASCII characters to encode graphic screen coordinates. Certain control characters are used as commands.

To enable the terminal to respond properly to 4010-style graphic commands, issue the commands

```
!GRAPHIC 1,35<CR>
!SHRINK<CR>
```

These set up a graphics region which is correctly proportioned to hold a 4010-style graphics display. Specifically, the addressable graphic region is approximately 640X by 525Y, in workspace coordinates (1024X by 780Y in 4010 coordinates). (See the SHRINK command discussion earlier in this section.)

The following 4010-style commands from the host cause the terminal to change operating modes:

1. The GS command — places the terminal in 4010-style graph mode.

2. The US command — exits the terminal from graph mode and positions the cursor at the character cell containing the vector beam.

3. The ESC command — notifies the terminal that the next character should be interpreted as a command. This command has no effect if the terminal is in 4010-style graph mode.

4. The ESC-Form Feed command — when in US mode, erases the current graphics area.

5. The ESC-SUB command — gives 4010-compatibility with GIN mode. When the command is received from the host, the crosshairs are brought up for one sample. This sample is sent in 4010-format. This command is not recognized from the keyboard.

Addressing the Vector Beam

The vector beam is moved to a point in the graphics region by sending to the terminal the binary equivalents of the Y address and the X address (4010 coordinate addresses) of the point. Each binary equivalent is separated into two parts: the five most significant bits and the five least significant bits. The address 205Y,148X translates to 0011001101Y, 0010010100X (binary). The 0011001101Y becomes 00110 HiY and 01101 LoY; the 0010010100X becomes 00100 HiX and 10100 LoX. In graph mode, these bytes cause the beam to be moved to the 205Y,148X position in the graphics area. To be sent to the terminal these bytes must be encoded as ASCII equivalents. The 00110 HiY bit is encoded as an ASCII "&" symbol, which has binary representation 0100110. The first two bits, 01, instruct the terminal that this is a HiY address. The last five bits, 00110, form the HiY segment of the Y address 0011001101. 205Y,148X is encoded as "&M$T." Appendix C is a Coordinate Conversion Chart for encoding X and Y-coordinates as ASCII characters.

Graph Mode Memory

When an address is sent to the terminal, the HiY, LoY and HiX bytes are stored in a register. If the next address sent to the terminal repeats some of these bytes, they need not be retransmitted. LoX must always be sent, since the command is not executed until LoX is received. Even if the terminal leaves graph mode and reenters it later, these three bytes are retained. The following table shows which bytes must be sent in response to specific byte changes.

Table 9-1

4010-STYLE GRAPHICS REQUIRED BYTE TRANSMISSIONS

Bytes Which Change	Bytes which must be transmitted			
	Hi Y	Lo Y	Hi X	Lo X
Hi Y	#			#
Lo Y		#		#
Hi X		#	#	#
Lo X				#

When the terminal exits 4010 mode, the communications port is returned to the portion of display memory it was in before entering 4010 mode (workspace or monitor).

For a complete discussion of 4010-style graphics, see the 4010 Series documentation.

ALTERNATE CHARACTER FONTS

The graphics memory may be used to store alternate character fonts, defined by the user for special purposes.

With 32K of graphics memory (Option 26), the terminal can accommodate up to 16 different user-defined fonts, each containing up to 128 characters. Alternate character fonts are defined, symbol by symbol, with the SYMBOL command.

Symbol

(Requires Option 25 or 26)

Syntax

```
!SYMbol <number><font> [<value 1>][<value 2>]...[<value 14>]<CR>
```

where

<number> is an ADE number between 00 and 127, or the actual ASCII character.
<font> is an integer between 1 and 31, inclusive.
<value n> is an integer from 0 to 255.

The <number> parameter may be either a two or three digit ADE value or the single ASCII charcter. The <font> parameter must specify a character font for which graphics memory is installed. (The operator can discover which character fonts have graphics memory installed with the GTEST command, discussed in the System Status and Initialization section.) Each <value n> parameter defaults to zero.

Action

This command defines a symbol in character font <font>. The <number> parameter is the ASCII decimal equivalent of some ASCII character. When this ASCII character is entered in a field with the indicated font attribute, the symbol defined by this command is displayed.

The symbol is defined by specifying which dots in the 8 x 14 character cell matrix are lighted when the symbol is displayed. Each <value n> parameter is converted into an 8-bit binary equivalent. The zero/one pattern of this binary equivalent determines which of the eight dots in the n-th row of the character cell will be lighted when this character is displayed.

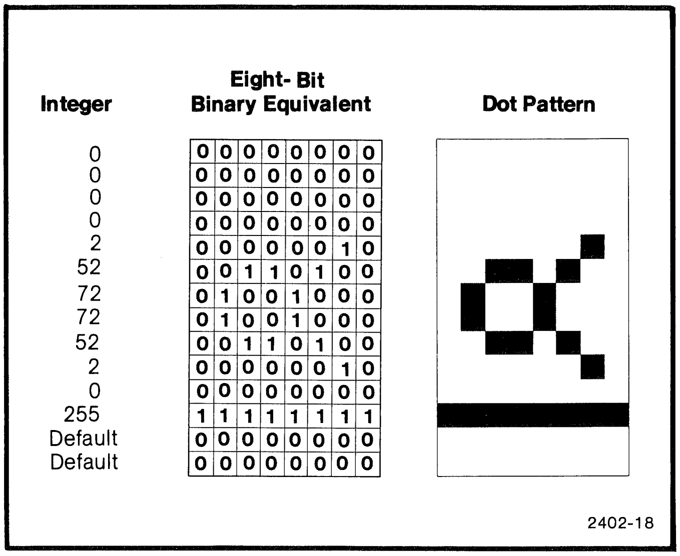
If fewer than 14 rows are specified, each of the unspecified parameters default to zero the remaining rows have no dots lighted and are displayed dark.

Example

The command

```
!SYM 97,30,0,0,0,0,2,52,72,72,52,2,0,255<CR>
```

defines character 97 of font 30. The number 97 is the ASCII Decimal Equivalent of the ASCII character a. Thus when the a character is entered in a field with font attribute 30, the symbol defined by this command is displayed. Figure 9-9 illustrates this symbol and how the SYMBOL command defines it.



Integer	Eight- Bit Binary Equivalent							
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	1	0
52	0	0	1	1	0	1	0	0
72	0	1	0	0	1	0	0	0
72	0	1	0	0	1	0	0	0
52	0	0	1	1	0	1	0	0
2	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	0
255	1	1	1	1	1	1	1	1
Default	0	0	0	0	0	0	0	0
Default	0	0	0	0	0	0	0	0

Figure 9-9. A User-Defined Symbol.

If one wishes to clear symbol 97 from user-defined font 30, the command

```
!SYM 97,30<CR>
```

is given. All rows in the character cell matrix are set to zero, and this symbol is displayed as a space, with no matrix dots turned on.

DFONT (Delete Font)

(Requires Option 25 or 26)

The graphics memory used to store symbol definitions in a user-defined character font can be cleared of these definitions and released for another use by giving the DFONT command.

Syntax

```
!DFOnt <font> <CR>
```

where <font> is an integer between 1 and 31, inclusive.

Action

This command deletes all symbol definitions in the specified font. The graphics memory used to store these definitions can now be used to store graphic information or another user-defined font.

Example

The command

```
!DFO 30<CR>
```

deletes from memory all symbol definitions in character font 30.

Section 10

PERIPHERALS

The 4025A supports the following peripherals:

- With Option 03 (RS-232 Peripheral Interface), the 4025A supports a TEKTRONIX 4642 Printer and other compatible printers.
- With Option 04 (GPIB Interface), the 4025A supports up to four TEKTRONIX 4924 Digital Cartridge Tape Drives and up to two TEKTRONIX 4662 or 4663 Interactive Digital Plotters.

Throughout this section the term "devices" is used to refer to one of the following: a peripheral device such as the printer or a tape unit, the host computer, the terminal monitor, or the terminal workspace. These devices are specified by device mnemonics as follows:

Device	**Mnemonic**
Printer	P or PR
Tape Units 1-4	TA1-TA4 (TA1 may be shortened to T)
Plotters 1,2	PL1, PL2
Terminal Monitor	M or MO
Terminal Workspace	W or WO
Host Computer	H or HO

INITIALIZING THE TERMINAL FOR PERIPHERAL COMMUNICATIONS

Before the terminal can copy file information from one device to another, it must be correctly informed about the status of the various peripheral devices attached to it. SET commands are used to initialize the termial for communicating with peripheral devices. The SET command should be given for those and only those peripheral devices present and powered up.

SET

(Requires Option 03 or 04)

Syntax

```
!SET <device><parameter>[<parameter>]<CR>
```

where

<device> is a one to three-letter device mnemonic, and <parameter> is a parameter setting for the indicated device. The form of each <parameter> depends on the peripheral device specified.

CAUTION

Do not attempt to set parameters for devices which are not attached to the terminal and powered up. Giving such a SET command may disable all communications to the terminal and require the terminal to be RESET.

SETting the Printer Parameters

(Requires Option 03)

The terminal comes from the factory set for communicating with a TEKTRONIX 4642 Printer. There are two parameters, however, which can be set to allow the terminal to communicate with printers other than the 4642 Printer.

- Some printers recognize the ASCII form feed character, <FF>, as a signal to begin a new page. Other printers do not; on such printers a series of line feeds (<LF>s) must be sent to begin a new page. The terminal can be set to send either the ASCII form feed (<FF>) character or the proper number of ASCII line feed (<LF>) characters to cause the printer to begin a new page.

- The "carriage return, line feed" mechanical operation which the printer uses to begin a new line is relatively slow, compared to terminal data transmission speeds. The terminal has a "printer delay" parameter which can be set. After the terminal sends a <CR>, <LF>, or <FF> to the printer, it waits the length of time specified by the printer delay parameter before sending another character. This gives the printer time to complete its mechanical functions before new text has to be printed.

The command which SETs the terminal for printer communications has the following form:

```
!SET PR [F | L][<delay>]<CR>
```

where:

- F stands for "form feed," and instructs the terminal to use the <FF> character as the page separator.
- L stands for "line feed," and instructs the terminal to replace any <FF> character to the printer by the number of <LF>s required to begin a new page.
- <delay> gives the printer delay. If <delay> is a positive integer, after the terminal sends a <CR>, <LF>, or <FF> to the printer, it waits <delay> tenths of a second before sending the next character. If <delay> = 0, the terminal communicates with the printer using "flagged simplex protocol." This means that after the terminal has sent a <CR>, <LF>, or <FF> to the printer, it waits for the RS-232 DTR (Data Terminal Ready) signal to become true before sending the next character.

Examples

!SET PR F 3<CR>	Instructs the terminal to use a form feed character (<FF>) as the page separator; when the printer receives a <FF>, it begins a new page. This command also sets the printer delay to 0.3 seconds; after sending a <CR>, <LF>, or <FF> to the printer, the terminal waits 0.3 seconds before sending another character.
!SET PR L<CR>	Instructs the terminal that the printer does not treat a <FF> character as the page separator. The terminal replaces a <FF> with the number of <LF>s required to begin a new page.
!SET PR 0<CR>	Instructs the terminal to communicate with the printer using flagged simplex protocol. After sending a <CR>, <LF>, or <FF>, the terminal waits for a DTR (Data Terminal Ready) signal from the printer before sending another character.

SETting the Tape Unit Parameters

(Requires Option 04)

To prepare the terminal to communicate with a 4924 Digital Cartridge Tape Drive (hereafter referred to as a "tape unit"), three parameters must be set.

- Since the terminal can have up to four tape units connected to it, each tape drive is numbered: tape unit 1, tape unit 2, tape unit 3, or tape unit 4.
- The 4924 is a GPIB device; that is, the terminal communicates with it using a GPIB (General Purpose Interface Bus). Each tape unit has two GPIB addresses: a receive (listen) address and a transmit (talk) address. Since a tape unit's transmit address is always numerically just one larger than its receive address, only the receive address must be set. The GPIB (receive) address of a tape unit must be set to an even number between 2 and 28, inclusive. This address must be physically set by switches on the back of the tape unit itself. But the terminal must also be SET to send messages destined for the tape unit to the proper GPIB address. This must be done **each time the terminal is turned on!** (See the 4025A Operator's Manual Peripheral Devices section for operating procedures.)
- The tape unit can record information in one of two formats. One format is compatible with the TEKTRONIX 4051 Graphic System internal tape drive. This is the format normally used to store file information for the terminal.

  The other format is compatible with the TEKTRONIX 4923 Digital Cartridge Tape Recorder. This format should be used only if you must exchange tape cartridges with a 4923.

The command which initializes the terminal for communicating with a tape unit has the following format:

```
!SET <device><address>[4051|4923]<CR>
```

where:

- <device> is one of the following: {TA1|TA2|TA3|TA4}. This identifies the given device as tape unit 1, tape unit 2, etc.
- <address> is an even number from 2 to 28, inclusive. This specifies the GPIB address assigned to the tape unit in all GPIB communications.

The default [4051|4923] setting is 4051; if this parameter is not specified, 4051-compatible format is assumed. The TA1 parameter may be abbreviated to T, but TA2, TA3, and TA4 may not be abbreviated.

A separate SET command must be given for each tape unit powered up and attached to the terminal. If only one tape unit is present, it must be designated TA1; if two tape units are present, they must be designated TA1 and TA2. The TA3 and TA4 designations are not used unless three or four tape units, respectively, are present.

Examples

```
!SET TA1 8 4051<CR>
!SET T 8<CR>
```

Instructs the terminal that tape unit 1 is present at GPIB address 8, and instructs the terminal to write data on tape unit 1 in 4051 compatible format. The GPIB address for a tape unit must be even. (Requires Option 04)

```
!SET TA2 10 4923<CR>
```

Instructs the terminal that tape unit 2 is present at GPIB address 10, and instructs the terminal to write data on this tape unit in 4923 compatible format. (Requires option 04)

SETting the Plotter Parameters

(Requires Option 04)

To prepare the terminal to communicate with a 4662 or 4663 Interactive Digital Plotter (hereafter referred to as the plotter), two parameters must be set.

- Since the terminal may have two plotters attached to it, each plotter present must be numbered: plotter 1 or plotter 2. If only one plotter is present, it must be designated plotter 1.
- Since the plotter is a GPIB device, it must be assigned a GPIB address. This address must be set physically by switches on the plotter; in addition, the terminal must be instructed to send information to the proper GPIB address. The plotter GPIB address may be any integer from 1 to 30 inclusive. *It must not, however, be a tape unit address plus one,* since this would duplicate the tape unit's transmit address. (See SETting the Tape Unit Parameters earlier in this section.)

The command which initializes the terminal to communicate with a plotter has the following format:

```
!SET <device><address><CR>
```

where:

- <device> is PL or PL1 (for plotter 1), or PL2 (for plotter 2).
- <address> is an integer from 1 to 30, inclusive; this integer specifies the GPIB address of the plotter.

If two plotters are present, a separate SET command for each plotter is required.

NOTE

If switch settings on the back of the plotter are changed while the plotter is powered on, these switches are not read by the plotter until power is cycled. If you change the plotter's address switches, go through the entire GPIB power up procedure. (See 4025A Operator's Manual, the Peripheral Devices section.)

Example

!SET PL1 15<CR>	Instructs the terminal that plotter 1 is present at GPIB address 15. This must agree with the address switch settings on the plotter. GPIB addresses are specified for those, and only those, devices present and powered up on the GPIB. (Requires Option 04)

PERIPHERALS

The PERIPHERALS command allows you to examine the terminal settings for communicating with peripheral devices.

Syntax

```
!PERipherals [<device>]<CR>
```

where <device> specifies a non-GPIB device on which the peripheral settings are to be listed. If <device> is not specified, it defaults to M (monitor).

Action

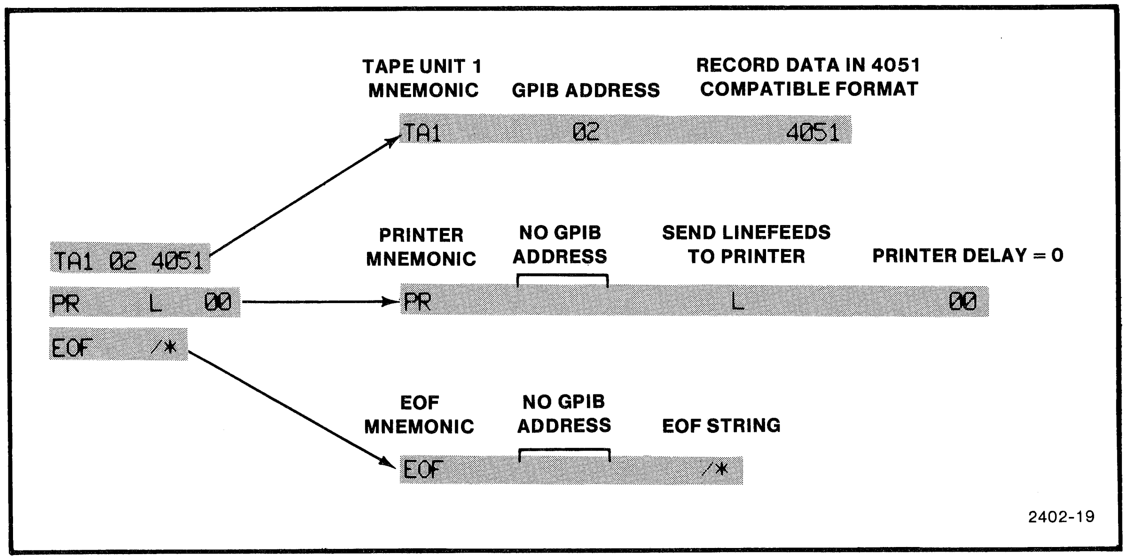
This command causes the terminal to generate a peripherals data list. For each device attached to the terminal and powered up, this list gives the <device> parameter (explained in the SET discussion), the GPIB address (this field is blank for the printer), and a data field listing the parameter settings for that device (explained in the SET command discussion).

The last line in the peripherals data list gives the EOF (end-of-file) string. (Setting the end-of-file string with the EOF command is discussed in System Status and Initialization.)

Example

!PER M<CR> !PER<CR>	Outputs a peripherals data list to the monitor.

A sample peripherals data list is shown in Figure 10-1.



```
TA1 02 4051
PR     L   00
EOF     /*
```

Figure 10-1. Peripherals Data List.

REPORT and Peripherals

The REPORT command has the following syntax:

```
!REPort <device><CR>
```

This command can be given only from the computer, not from the keyboard. The Host Programming section discusses the REPORT command and the format of the ANSwer sent to the host for device 00 (System Status Block) and device 01 (workspace cursor). The terminal can also report the status of each peripheral device, whether or not the given peripheral is present (attached to the terminal and powered up on the GPIB). This allows the program to investigate which peripherals are present at a given time and branch or modify instructions accordingly.

The peripherals have the following <device> numbers assigned:

<device>	**Peripheral(s)**
04-07	Tape Units 1-4, respectively
12,13	Plotters 1, 2, respectively
14	Printer

REPORTing the Tape Unit

When the command:

```
!REPort 0n<CR>
```

is given, and n is chosen from {1, 2, 3, 4}, the status of the designated tape unit is reported to the computer. This report has the following format:

```
!ANS 0n,<p1>,<p2>,<p3>;
```

where:

- <p1> = 1 if the tape unit is present; 0, if not.

- <p2> is a two-digit decimal value indicating the last tape error code. (See Table 10-1.)

- <p3> (4 bytes) = 4051 or 4923, indicating the format in which information is to be written on the tape.

Table 10-1

TAPE ERROR CODES

Code	Meaning
01	Domain error or invalid argument
02	File not found
03	Mag tape format error
04	Illegal access
05	File not open
06	Read error
07	No cartridge inserted
08	Over-read
09	Write-protected
10	Read-after-write error
11	End of medium
12	End of file

REPORTing the Plotter

When the command:

```
!REP 12<CR>
```

is given, the status of plotter 1 is reported to the computer. This report has the following format:

```
!ANS 12,<p1>,<p2>;
```

where:

- <p1> = 1 if the plotter is present; 0, if not.
- <p2> (6 bytes) is the ASCII integer value of plotter status word 0.

The command:

```
!REP 13<CR>
```

causes a similar report for plotter 2 to be sent to the computer.

REPORTing the Printer

When the command:

```
!REP 14<CR>
```

is given, the status of the printer is reported to the computer. This report has the following format:

```
!ANS 14,<p1>,<p2>,<p3>;
```

where:

- <p1> = 1 if the printer is present; 0, if not.
- <p2> = L if the line feed option is used, F if the form feed option is not used. (See the SETting the Printer Parameters discussion in this section.)
- <p3> (3 digits) is the ASCII integer value of the printer delay. (See the SETting the Printer Parameters discussion in this section.)

COMMUNICATING WITH PERIPHERALS

Commands which enable the terminal to communicate with peripheral devices are the ALLOCATE, DIRECTORY, and KILL commands, for communicating with a tape unit; the PASS command, used to communicate with a plotter; and the COPY command, used to copy files from one device to another.

ALLOCATE

(Requires Option 04 and a 4924 Tape Unit)

Before information can be recorded on a tape, files must be created on the tape to hold the information. This is done by using the ALLOCATE command.

Syntax

```
!ALLocate <device><beg file><number><size><CR>
```

where:

- <device> is a device mnemonic (T[A1], TA2, TA3, or TA4) which specifies the tape unit used to record information.
- <beg file> is a non-negative integer which specifies the number of the first file to be created.
- <number> is a positive integer which specifies the number of files to be created.
- <size> is a positive integer which specifies the number of eight-bit bytes which each newly created file is to contain. Each tape cartridge can store approximately 250K bytes of information.

Action

This command creates new files on a tape inserted in the tape unit specified by the <device> parameter.

<beg file> =0. If the tape has not previously been used to record information, <beg file> must be set to zero. This causes the tape to be properly initialized before a file structure is recorded on it. If the tape has already been used to record information, setting <beg file> to 0 destroys all information previously recorded on the tape, including the file structure marked on the tape; then the tape is reinitialized. In either of these cases, new files 1 through <number> are created. Each new file contains enough space to store <size> eight-bit bytes of information.

<beg file> positive. If <beg file> is positive, this command creates <number> consecutive new files on the tape. The first new file created is file number <beg file> and each new file contains enough space to hold <size> eight-bit bytes of information.

Examples

INITIALIZING AN UNMARKED TAPE

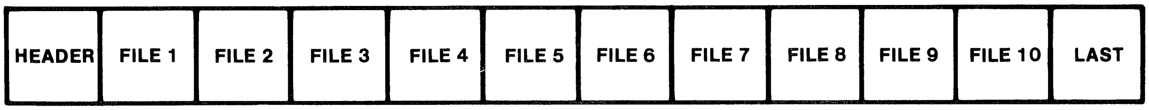
`!ALL TA1 0,2,5000<CR>`
`!ALL T 0,2,5000<CR>`
	Initializes the unmarked tape in tape unit 1 and creates two files (files 1 and 2) of 5000 bytes each. (If the tape has already been marked, this command destroys all old information on the tape.)

ALLOCATING FILE SPACE ON A MARKED TAPE

`!ALL TA1 1,2,5000<CR>`
`!ALL T 1,2,5000<CR>`
	Creates two new file on tape unit 1, beginning with file 1. Each new file contains 5000 bytes.

`!ALL TA1 7,4,8000<CR>`	Creates four new files on tape unit 1, beginning with file 7. Each new file contains 8000 bytes.

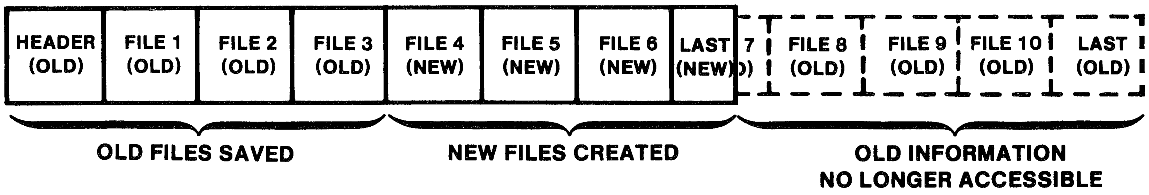
In addition to the <number> new files created, an ALLOCATE command attaches a file called LAST immediately after the last newly created file. The LAST file is always 768 bytes long. It marks the logical end of the file structure on the tape. If new files are allocated in the middle of an existing file structure, all the old information on the tape from file <beg file> to the end of the tape is lost, even if the newly ALLOCATEd space is shorter than previously ALLOCATEd space. Suppose you have 10 files of 5000 bytes each on the tape in tape unit 1:



The command

```
!ALL TA1 4,3,5000<CR>
```

creates new files 4, 5, and 6 (destroying the old files 4, 5, and 6) and attaches a LAST file immediately following file 6:



The tape now contains six files. Even though old files 8-10 and part of old file 7 are still magnetically recorded on the tape, this information is no longer accessible.

DIRECTORY

(Requires Option 04)

When information has been stored on a tape, it will be necessary at times to examine the file structure on the tape (perhaps to recall how many files have already been created). The DIRECTORY command allows us to do this.

Syntax

```
!DIRectory <tape device> [<output device>]<CR>
```

where:

- <tape device> specifies a tape unit.
- <output device> specifies a non-GPIB device. If this parameter is not specified, it defaults to M (monitor).

Action

This command outputs file header information stored on the tape in the tape unit specified by <tape device>. The information on this tape must be recorded in 4051-compatible format. (Files recorded in 4923-compatible format do not contain file header information.) This information is recorded on the device specified by the <output device> parameter. This output device must be a non-GPIB device. Each file header lists the file number, the file type, and the length of the file.

There are three types of files: a NEW file is one which has been marked on the tape, but no information has yet been recorded in it; a file with information recorded in it is an ASCII DATA file; the LAST file marks the logical end of the tape file structure.

Even though file lengths are ALLOCATEd in terms of bytes, the DIRECTORY command lists file lengths in blocks. In 4051-compatible format, a block consists of 256 bytes.

Examples

Suppose you have just given the command

```
!ALL TA1 0,5,5000<CR>
```

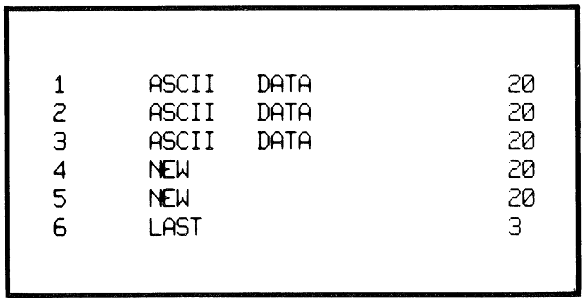
This initializes the tape and creates five files of 5000 bytes each, as well as a sixth file, LAST. Suppose data is entered in files 1, 2 and 3, and one of the following commands is given:

```
!DIR TA1 M<CR>
```

or

```
!DIR T<CR>
```

The following list is displayed in the monitor:



```
1      ASCII   DATA              20
2      ASCII   DATA              20
3      ASCII   DATA              20
4      NEW                       20
5      NEW                       20
6      LAST                      3
```

Note that a LAST file is always three blocks (768 bytes) long.

The command:

```
!DIR TA2 P<CR>
```

prints tape unit 2 file headers on the printer.

KILL

When a (4051-compatible) file is first created, its file header reads NEW, meaning that the file exists but no information is recorded in it (except the file header information). When information is stored in that file, its file header is changed to ASCII DATA.

An ASCII DATA file can be restored to its NEW status by the KILL command.

Syntax

```
!KILI <tape unit.file number> <CR>
```

Example

If you wish to restore file 6 on the tape in tape unit 1 to its NEW status, give the command:

```
!KIL TA1.6<CR>
```

This restores the file header for that file to NEW and any information stored in the file is lost.

PASS

(Requires Option 04)

The terminal can display graphs and text on a TEKTRONIX 4662 Interactive Digital Plotter. The terminal has a vocabulary of commands for creating graphic displays. (See the Graphics section.) The plotter also has a vocabulary of commands for creating graphic displays. This plotter language includes more graphic capabilities than the terminal language, and the formats of the two languages are different. Table 10-2 gives a summary of the plotter-command language.

Ultimately, any command to which the plotter responds *must* be in plotter command language. The terminal is designed so that the user can specify graphic commands destined for the plotter in either of two ways.

1. Create a file of plotter-language commands and send that file to the plotter via the terminal.

   This method has the advantage of using the full range of plotter commands; however, it has two disadvantages. First, such a command file can be used only for drawing graphs on the plotter. Since the terminal does not understand plotter-language commands, this file is meaningless to it. Second (and more serious), the plotter language uses the ASCII<ETX> (end-of-text) character. Being a control character, it will not be displayed or inserted in the workspace unless the terminal is in snoopy mode. The <ETX> character is also used frequently in communications with the host computer. Transmitting a plotter-language command file containing <ETX>s to or from some computers may cause unintended results.

2. Create a command file of graphic commands and send that file to the plotter, instructing the terminal to translate them into plotter commands.

   This method has the advantage that the command file thus created can be used to display the same graph both in the workspace and on the plotter. Such a file is easily transmitted to and from the computer with no troublesome control characters.

   This method has a disadvantage, however. Since the plotter's vocabulary of graphic commands is larger and more versatile than that of the terminal, a command file using only terminal commands cannot use the full range of the plotter's graphic capabilities.

To send 4025A-style command files to the plotter, use the COPY command with the /P switch setting. (See the COPY discussion later in this section.) To send plotter-style command files containing <ETX>s, use the PASS command.

The PASS command allows you to transmit control characters to and from the computer by encoding them as ASCII Decimal Equivalents.

Syntax

```
!PASs <string> <CR>
```

where <string> can be:

- A delimited ASCII string.
- One or more ADE values.
- A combination of the above.

Table 10-2 gives a summary of the plotter command vocabulary. Table 10-3 illustrates how various plotter commands are transmitted using the PASS command.

Table 10-2

PLOTTER LANGUAGE COMMANDS

Command	Action
H<CR> or H<ETX>	**Home.** Moves the pen to the upper left corner of the plotting area.
M 50,75<CR> or M50,75<ETX>	**Move.** Moves the pen to the point (50,75) in the plotter system of coordinates. *NOTE* *The plotter's coordinate system is not the same as the terminal's coordinate system. The plotter's X-axis always runs from 0 to 150, and its Y-axis runs from 0 to 100.*

Table 10-2 (cont)

PLOTTER LANGUAGE COMMANDS

Command	Action
D 100,50<CR> or D100,50<ETX>	**Draw.** Draws a line from the current pen position to the point (100,50) in the plotter's system of coordinates.
D 100,50 0,0 50,10<CR> or D100,50,0,0,50,10<ETX>	**Draw.** Draws a line from the current pen position to the point (100,50); from there, to the point (0,0); and from there, to (50,10).
PThis is a test.<ETX>	**Print.** Prints on the plotter, starting at the current pen position, the message "This is a test."
S 1.5,3.0<CR> or S1.5,3.0<ETX>	**Alpha Scale.** Sets the size of each character cell to 1.5 graphic display units in the X-direction and 3.0 graphic display units in the Y-direction. (The "graphic display unit" is a measure of length used by the plotter. In the X-direction, it is 1/150 the length of the plotting area; in the Y-direction, it is 1/100 the height of the plotting area.)
R 10<CR> or R10<ETX>	**Alpha Rotate.** Sets the angle at which alphanumeric characters are printed on the plotting surface. Characters printed after this command is executed slant upwards at ten degrees with respect to the positive X-axis.
F 2<CR> or F2<ETX>	**Alpha Font.** Selects printing font number 2 from among the plotter's seven fonts.
A<CR> or A<ETX>	**Alpha Reset.** Resets the alphaunumeric printing parameters (Alpha Scale, Alpha Rotate, Alpha Font) to their default values.
T 0<CR> or T0<ETX>	**Prompt Light.** Turns off the PROMPT light on the plotter's front panel.
T 1<CR> or T1<ETX>	**Prompt Light.** Turns on the plotter's PROMPT light.

NOTE

The coordinates in the 4025A-language VECTOR command differ from those in the plotter-language MOVE and DRAW commands. In translating VECTOR commands, the terminal assumes a graphics area with a maximum X-coordinate of 639 and maximum Y-coordinate of 479 is to be mapped onto a plotter work area with maximum X-coordinate of 150 and maximum Y-coordinate of 100.

Table 10-3

TRANSMITTING PLOTTER COMMANDS USING PASS

Command Name	Plotter Language	4025A Language
HOME	H< CR> H< ETX>	!PASS "H",13< CR> !PASS"H",3< CR>
MOVE	M 50,75< CR> M50,75< ETX>	!PASS "M 50,75",13< CR> !PASS "M50,75",3< CR>
DRAW	D100,50< ETX> D100,50,0,0< ETX>	!PASS "D100,50",3< CR> !PASS "D100,50,0,0",3< CR>
MOVE, followed by DRAW	M50,75 D100,100< ETX>	!PASS "M50,75" "D100,100",3 !VEC 213,30,427,479< CR>
PRINT	PThis is a test.< ETX>	!PASS"PThis is a test.",3< CR> !STRING "This is a test."< CR>
ALPHA SCALE	S1.5,3.0< ETX>	!PASS "S1.5,3.0",3< CR>
ALPHA ROTATE	R10< ETX>	!PASS "R10",3< CR>
ALPHA FONT	F2< ETX>	!PASS "F2",3< CR>

Table 10-3 (cont)

TRANSMITTING PLOTTER COMMANDS USING PASS

Command Name	**Plotter Language**	**4025A Language**
ALPHA RESET	A<ETX>	!PASS "A",3<CR>
PROMPT LIGHT	T0<ETX> T1<ETX>	!PASS "T0",3<CR> !PASS "T1",3<CR>

NOTE

Only PRINT requires the <ETX> character. All other commands take <ETX> or <CR>.

4025A COPY

(Requires Option 03 or 04)

The terminal with Options 03 or 04 can copy from the workspace or from a host file to a TEKTRONIX 4642 Printer.

Since the printer cannot print rulings, all attribute codes will be ignored and all rulings will be converted to asterisks when the terminal sends form data to the printer.

When the terminal copies a file from the computer to the printer, it uses the end-of-file string to know when to stop copying. The user specifies the end-of-file string. This string can be any ASCII string of up to ten characters. The host file should always end with an end-of-file string, and the terminal must be set to recognize this particular string as indicating the end of a file. The EOF command instructs the terminal to recognize a particular string as the end-of-file string. (See the discussion of the EOF command in the System Status and initialization section.)

Files are copied from the computer or the workspace to the printer with the COPY command.

Syntax

```
!COPY <source> [<switches>][<destination>]<CR>
```

Action

This command copies the information contained in <source> to <destination>. If the <switches> parameter is present, the terminal receives or transmits information according to certain conventions determined by the value of <switches>. The <source> and <destination> parameters are shown in Table 10-4.

Table 10-4

COPY PARAMETERS

Device	Mnemonic Used As Parameter	Used As <source>	Used As <destination>
Workspace	W or WO	X	X
Monitor	M or MO		
Host Computer	H or HO	X	X
Printer	P or PR		X
4662 or 4663 Plotter 1 and 2	PL1, PL2	X	X
A file on a given tape unit	TAn.k	X	X

A particular file on a tape is designated TAn.k (see Table 10-4). In this notation, n is the number of a tape unit ($1 \leq n \leq 4$) and k is the number of a file on the given tape unit (e.g., TA1.3 or TA3.15). The TA1 mnemonic can be shortened to T; for example, TA1.3 can be written T.3.

If <destination> is not specified, it defaults to W (Workspace).

Examples

!COP W H<CR>	Copies the contents of the workspace to the host, with EOF string at the end.
!COP W TA1.3<CR> !COP W T.3<CR>	Copies the workspace contents to file 3 of tape unit 1.
!COP TA3.5 W<CR> !COP TA3.5<CR>	Copies file 5 of the tape in tape unit 3 to the workspace.
!COP TA3.5 P<CR>	Copies file 5 of tape unit 3 to the printer.
!COP TA2.15 PL1<CR>	Copies file 15 of tape unit 2 to plotter 1.

NOTE

The COPY operation is non-destructive. The command !COP TA1.3 W<CR> copies the contents of file 3, tape unit 1 to the workspace, leaving that information still stored in file 3 of tape unit 1.

The <switches> parameter consists of one or more slashes (/), each followed by a single letter. Each letter serves as a "switch" which, if present, instructs the terminal to receive or transmit information in a certain way. Each switch is given for a specific purpose and is, strictly or loosely, associated with a specific <source> or <destination>. The switches and their uses are summarized in Table 10-5.

Table 10-5

4025A COPY SWITCHES

Switch	Use
/N	When <source> is the workspace, this switch instructs the terminal to ignore all attribute codes stored in the workspace display list and convert all ruling characters to asterisks. In this way, a form containing ruling characters can be copied to the printer.

Table 10-5 (cont)

4025A COPY SWITCHES

Switch	Use
/N	When <destination> is the workspace, this switch instructs the terminal to treat all information coming from <source>, including the command character, as text to be displayed. This allows the terminal operator to display a file containing commands in the workspace for examination and modification. Setting this switch is equivalent to pressing COMMAND LOCKOUT while the terminal receives text from the host computer. Once the file is displayed in the workspace, however, the terminal again recognizes the command character. To send such a file from the workspace to some other <destination> one can do either of two things: 1. Initiate a file transfer from the workspace (COPY or, if <destination> is the host computer, SEND). 2. If the host is going to echo the data back, simply change the command character to a symbol which does not appear in the displayed file. Then COPY or SEND as usual.
/U	When <source> is the workspace and the workspace holds a form, this switch instructs the terminal to copy data in the **unprotected fields only**. If a form is being filled out repeatedly and you wish to store just the data in the form, this switch is used. If the workspace holds a form and this switch is not set, a COPY operation copies the entire form, including attribute codes and data in protected fields. If the workspace does not contain a form (no attribute codes inserted in the display list), the /U switch has no effect.
/D	When <source> is the host computer and <destination> is neither workspace nor monitor, this switch instructs the terminal to display the copied file in the workspace. Without this switch set, a file copied from the host to file TA1.3, for example, would not be displayed on the screen.
/P	When a file containing graphic commands is copied with the /P switch set, the terminal translates all VECTOR, RVECTOR, STRING, and PASS commands into plotter-command language. This enables the same file to be used to create graphs on both the terminal and the plotter.

Examples

!COP W H<CR>	Copies the contents of the workspace to the host and includes an EOF at the end of the communication. (Requires Option 36)
!COP W/N P<CR>	Copies the workspace contents to the printer, ignoring attribute codes and converting all ruling characters to asterisks. (Requires Option 03)
!COP W/U TA1.4<CR>	If the workspace holds a form, copies data from the **unprotected fields only** to file 4 of tape unit 1. (Requires Option 04)
!COP H/D TA2.5<CR>	Copies the host file to file 5 of tape unit 2, displaying this file in the workspace as it is copied. (Requires Option 04)
!COP TA1.7/P PL1	Copies file 7 of tape unit 1 to the plotter, translating all VECTOR, RVECTOR, STRING, and PASS commands into plotter-command language. (Requires Option 04)

It is possible to set more than one switch in a COPY command. The command

```
!COP H/D/P PL1<CR>
```

copies the host file to plotter 1, displaying this file in the workspace and translating terminal graphic commands into plotter-command language.

Auto-Incrementing the Tape Unit

When a particular file on a tape unit is designated in a COPY command, the terminal remembers that file number until it is replaced by another file number (or until the terminal is RESET or powered off). If the file number is omitted in a subsequent COPY command, the terminal automatically increments the file number in its memory by one and copies to or from that file on the tape.

This feature is useful in text editing or form fillout. Suppose the operator has created a form in the workspace and stored the form on tape unit 1 with the command

```
!COP W TA1.1<CR>
```

Suppose the PT (Pad Terminator) key has been programmed to give the command

```
!COP W/U TA1!ERA W<CR>
```

The operator fills out the form and presses PT. The terminal stores the data from the unprotected fields in the next available file (file 2) of tape unit 1 and erases the workspace. (With the terminal in form fillout mode, only the data in the unprotected fields is erased.) The operator fills out the form again, with different data, and again presses PT. Again the terminal stores the data from the unprotected fields in the next available file (now file 3) of tape unit 1 and erases the blanks of the form. The operator proceeds in this way as long as necessary.

As another example, suppose successive pages of text to be edited are stored in files 12-27 of tape unit 2. To get the first page into the workspace, give the command

```
!COP TA2.12<CR>
```

To get each succeeding page, give the command

```
!COP TA2<CR>
```

Copying the Workspace to the Plotter

If the workspace holds a graph, it is tempting to try copying this graph directly to the plotter. This **cannot** be done. The workspace display list does not contain sufficient information to translate the graphic information displayed into commands which can recreate the graph. If the terminal is commanded to copy the workspace to the plotter, graphic information does not copy. (This is similar to the SEND command.) To obtain graphs, one must store somewhere (on a tape or in the host) a command file containing the necessary commands to recreate the graph. These cannot be derived from the graphic display alone.

COPYING ON A HARD COPY UNIT

A TEKTRONIX 4611 or 4631 Hard Copy Unit can be used to make copies of the workspace, the monitor, or the screen.

This 4611 or 4631 Hard Copy Unit produces pages of copy approximately 8-1/2 inches by 11 inches in size. These copies can show whatever can be displayed on the screen: text, control characters in snoopy mode, rulings, alternate character fonts, visual attributes (except blinking), and graphs. One or several 53-line "pages" from the workspace or the monitor can be copied.

Hard copies are made on the 4611 or 4631 Hard Copy Unit with the HCOPY command.

HCOPY (Hard Copy)

Syntax !HCOpy [<count>][Workspace|Monitor|Screen]<CR>

where <count> is a positive integer. If <count> is not specified, it defaults to one.

Action

If M (monitor) or W (workspace) is specified, this command copies <count> "pages" from the specified scroll to a TEKTRONIX 4611 or 4631 Hard Copy Unit. The copy begins with the first visible line in that scroll. Each "page" of copy continues until it includes 53 lines of text or until an ASCII <FF> character appears in column 1 of a line. The line of text containing such a form feed is not copied. If the specified scroll contains fewer than <count> pages, only the number of pages in the scroll is copied. If one attempts to make a hard copy of a blank scroll, one (blank) page of hard copy will be produced.

If S (screen) is specified, <count> is ignored and one copy of the visual screen display, both workspace and monitor windows, is made.

If the HCOPY command comes from the computer and neither W nor M nor S is specified, pages are copied from whichever scroll receives text from the computer.

If the HCOPY command is typed on the keyboard and neither W nor M nor S is specified, pages are copied from whichever scroll receives text from the keyboard.

Examples

!HCO 3 W<CR>

Copies three "pages" from the workspace to the Hard Copy Unit. The first page copied begins with the first line visible in the workspace.

NOTE

This means one copy each of three consecutive pages in the workspace scroll, not three copies of the same page. If the workspace contains only one page, then only one sheet of copy is produced.

!HCO M<CR>

Copies one page from the monitor to the Hard Copy Unit. This page begins with the first line visible in the monitor.

!HCO S<CR>

Copies all information displayed on the screen to the Hard Copy Unit.

!HCO<CR>

If this command comes from the computer, copies a page from the scroll which receives text from the computer.

If this command is typed on the keyboard, copies a page from the scroll which receives text from the keyboard.

Appendix A

MEMORY CONSIDERATIONS

It is possible for the terminal to use up all of its display memory. Likewise, the terminal may use up all of its graphics memory. The comments in this appendix should help the programmer judge how much of each type of information can be sent to the terminal before running out of memory.

DISPLAY MEMORY

On the terminal screen, a full line of text (a character displayed in every column) plus ten attribute codes uses 112 bytes. A full screen of such lines (34 lines X 112 bytes/line) uses about 3800 bytes. As a rule of thumb, then, you get about one full screen of display for every 4K bytes of display memory. Usually, of course, one does not use the full 80 columns of a line for display. A rough calculation of line length will give the proper adjustment factor. For example, if the program uses roughly 50% of each line for display, a 16K byte display memory will store approximately:

$$\frac{16\text{K bytes} \times 1\text{ screen}}{4\text{K bytes}} \times 2 = 8\text{ 34-line screens}$$

The workspace and the monitor both use memory out of the same "pool" of display memory. When the terminal has used most of its display memory and you attempt to display more information, the result depends on which scroll is receiving information.

If the terminal runs low on memory while you are sending information to the workspace, the terminal bell rings as a warning to the operator, and the terminal overprints a portion of the current line with incoming data. If information continues to come, the terminal soon refuses to print and the cursor sticks at its current location.

If the terminal runs low on memory while you are sending information to the monitor, the cursor simply sticks at its current location and the terminal refuses to print new information. The terminal still processes a carriage return, however, and enough memory is saved to give at least one command. (If the monitor has scrolled information up past the top of the monitor window, the terminal discards this information as needed, line by line. In this case, you may keep sending information to the monitor, and the terminal will keep discarding scrolled up information.)

An applications program may keep track of the amount of unused display memory by occasionally giving the command

```
!REP00<CR>
```

This command causes the terminal to return a report to the computer in the following format:

```
!ANS 00,<p1>,<p2>;
```

where <p1> is a four-digit decimal number specifying the number of unused blocks of display memory. (A block consists of 16 8-bit bytes.) When <p1> falls below a given level, the program can instruct the terminal to erase information by sending an ERASE, DLINE, or DCHAR command or,.if the information displayed is not needed, a WORKSPACE or MONITOR command. To recover display memory from the computer, you must give some command which erases text.

GRAPHICS MEMORY

The terminal can contain 16K or 32K bytes of graphics memory. The amount of graphics memory which a given graph requires depends on the density of information in the graph. The following is an estimate of how much graphics memory is required for graphic display. The term "pie chart" refers to a pie chart with 10 pieces. The term "graph" refers to a line graph of approximately the same density as the graph in Figure 9-6.

Table A-1

4025A GRAPHIC MEMORY CAPACITY

Amount of Graphic Memory	Display Capacity
16K bytes	2+ pie charts, or 2 line graphs
32K bytes	4 pie charts, or 3 to 4 line graphs

To recover graphics memory, use the ERASE G command.

Appendix B

THE ASCII CODE

Table B-1

ASCII CODE CHART

B7 B6 B5				Ø Ø Ø	Ø Ø 1	Ø 1 Ø	Ø 1 1	1 Ø Ø	1 Ø 1	1 1 Ø	1 1 1
BITS				CONTROL		HIGH X & Y GRAPHIC INPUT		LOW X		LOW Y	
B4	B3	B2	B1								
Ø	Ø	Ø	Ø	NUL 0	DLE 16	SP 32	0 48	@ 64	P 80	` 96	p 112
Ø	Ø	Ø	1	SOH 1	DC1 17	! 33	1 49	A 65	Q 81	a 97	q 113
Ø	Ø	1	Ø	STX 2	DC2 18	″ 34	2 50	B 66	R 82	b 98	r 114
Ø	Ø	1	1	ETX 3	DC3 19	# 35	3 51	C 67	S 83	c 99	s 115
Ø	1	Ø	Ø	EOT 4	DC4 20	$ 36	4 52	D 68	T 84	d 100	t 116
Ø	1	Ø	1	ENQ 5	NAK 21	% 37	5 53	E 69	U 85	e 101	u 117
Ø	1	1	Ø	ACK 6	SYN 22	& 38	6 54	F 70	V 86	f 102	v 118
Ø	1	1	1	BEL 7	ETB 23	' 39	7 55	G 71	W 87	g 103	w 119
1	Ø	Ø	Ø	BS 8	CAN 24	(40	8 56	H 72	X 88	h 104	x 120
1	Ø	Ø	1	HT 9	EM 25	) 41	9 57	I 73	Y 89	i 105	y 121
1	Ø	1	Ø	LF 10	SUB 26	* 42	: 58	J 74	Z 90	j 106	z 122
1	Ø	1	1	VT 11	ESC 27	+ 43	; 59	K 75	[91	k 107	{ 123
1	1	Ø	Ø	FF 12	FS 28	, 44	< 60	L 76	\ 92	l 108	|* 124
1	1	Ø	1	CR 13	GS 29	– 45	= 61	M 77	] 93	m 109	} 125
1	1	1	Ø	SO 14	RS 30	. 46	> 62	N 78	^ 94	n 110	~ 126
1	1	1	1	SI 15	US 31	/ 47	? 63	O 79	_ 95	o 111	RUBOUT (DEL) 127

* ¦ on some keyboards or systems.

Table B-2

ASCII CONTROL CHARACTERS

Mnemonic	Usual ASCII Abbrev.	Name of Character	Keys to Press
NU	NUL	Null	CRTL- @
SH	SOH	Start of Heading	CTRL-A
SX	STX	Start of Text	CTRL-B
EX	ETX	End of Text	CRTL-C
ET	EOT	End of Transmission	CTRL-D
EQ	ENQ	Enquiry	CTRL-E
AK	ACK	Acknowledgement	CTRL-F
BL	BEL	Bell	CTRL-G
BS	BS	Backspace	CTRL-H
HT	HT	Horizontal Tab	CTRL-I
LF	LF	Line Feed	CTRL-J
VT	VT	Vertical Tab	CTRL-K
FF	FF	Form Feed	CTRL-L
CR	CR	Carriage Return	CTRL-M
SO	SO	Shift Out	CTRL-N
SI	SI	Shift In	CTRL-O
DL	DLE	Data Link Escape	CTRL-P
D1	DC1	Device Control 1	CTRL-Q
D2	DC2	Device Control 2	CTRL-R
D3	DC3	Device Control 3	CTRL-S
D4	DC4	Device Control 4	CTRL-T
NK	NAK	Negative Acknowledgement	CTRL-U
SY	SYN	Synchronization Character	CTRL-V
EB	ETB	End of Transmission Block	CTRL-W
CN	CAN	Cancel	CTRL-X
EM	EM	End of Medium	CTRL-Y
SB	SUB	Substitute	CTRL-Z
EC	ESC	Escape	CTRL-[
FS	FS	Field Separator	CTRL-\
GS	GS	Group Separator	CTRL-]
RS	RS	Record Separator	CTRL-↑ CTRL- ∧ (CTRL-up arrow or CTRL-circumflex accent)
US	US	Unit Separator	CTRL-_ (CTRL-underscore)

Appendix C

4010-STYLE GRAPHICS CODES

Table C-1

4010-STYLE GRAPHICS CODE CHART

Low Order X										Low Order Y	
ASCII	DEC.	X or Y Coordinate								DEC.	ASCII
@	64	0	32	64	96	128	160	192	224	96	`
A	65	1	33	65	97	129	161	193	225	97	a
B	66	2	34	66	98	130	162	194	226	98	b
C	67	3	35	67	99	131	163	195	227	99	c
D	68	4	36	68	100	132	164	196	228	100	d
E	69	5	37	69	101	133	165	197	229	101	e
F	70	6	38	70	102	134	166	198	230	102	f
G	71	7	39	71	103	135	167	199	231	103	g
H	72	8	40	72	104	136	168	200	232	104	h
I	73	9	41	73	105	137	169	201	233	105	i
J	74	10	42	74	106	138	170	202	234	106	j
K	75	11	43	75	107	139	171	203	235	107	k
L	76	12	44	76	108	140	172	204	236	108	l
M	77	13	45	77	109	141	173	205	237	109	m
N	78	14	46	78	110	142	174	206	238	110	n
O	79	15	47	79	111	143	175	207	239	111	o
P	80	16	48	80	112	144	176	208	240	112	p
Q	81	17	49	81	113	145	177	209	241	113	q
R	82	18	50	82	114	146	178	210	242	114	r
S	83	19	51	83	115	147	179	211	243	115	s
T	84	20	52	84	116	148	180	212	244	116	t
U	85	21	53	85	117	149	181	213	245	117	u
V	86	22	54	86	118	150	182	214	246	118	v
W	87	23	55	87	119	151	183	215	247	119	w
X	88	24	56	88	120	152	184	216	248	120	x
Y	89	25	57	89	121	153	185	217	249	121	y
Z	90	26	58	90	122	154	186	218	250	122	z
[	91	27	59	91	123	155	187	219	251	123	{
\	92	28	60	92	124	156	188	220	252	124	¦
]	93	29	61	93	125	157	189	221	253	125	}
^	94	30	62	94	126	158	190	220	254	126	~
—	95	31	63	95	127	159	191	223	255	127	RUBOUT (DEL)
DEC. →		32	33	34	35	36	37	38	39		
ASCII →		SP	!	"	#	$	%	&	'		
		High Order X & Y									

INSTRUCTIONS: Find coordinate value in body of chart; follow that column to bottom of chart to find decimal value or ASCII character which represents the High Y or High X byte; go to the right in the row containing the coordinate value to find the Low Y byte, or go to the left to find the Low X byte. EXAMPLE: 200Y, 48X equals 38 104 33 80 in decimal code, and equals & h ! P in ASCII code.

Table C-1 (cont)

4010-STYLE GRAPHICS CODE CHART

Low Order X										Low Order Y	
ASCII	DEC.	X or Y Coordinate								DEC.	ASCII
@	64	256	288	320	352	384	416	448	480	96	`
A	65	257	289	321	353	385	417	449	481	97	a
B	66	258	290	322	354	386	418	450	482	98	b
C	67	259	291	323	355	387	419	451	483	99	c
D	68	260	292	324	356	388	420	452	484	100	d
E	69	261	293	325	357	389	421	453	485	101	e
F	70	262	294	326	358	390	422	454	486	102	f
G	71	263	295	327	359	391	423	455	487	103	g
H	72	264	296	328	360	392	424	456	488	104	h
I	73	265	297	329	361	393	425	457	489	105	i
J	74	266	298	330	362	394	426	458	490	106	j
K	75	267	299	331	363	395	427	459	491	107	k
L	76	268	300	332	364	396	428	460	492	108	l
M	77	269	301	333	365	397	429	461	493	109	m
N	78	270	302	334	366	398	430	462	494	110	n
O	79	271	303	335	367	399	431	463	495	111	o
P	80	272	304	336	368	400	432	464	496	112	p
Q	81	273	305	337	369	401	433	465	497	113	q
R	82	274	306	338	370	402	434	466	498	114	r
S	83	275	307	339	371	403	435	467	499	115	s
T	84	276	308	340	372	404	436	468	500	116	t
U	85	277	309	341	373	405	437	469	501	117	u
V	86	278	310	342	374	406	438	470	502	118	v
W	87	279	311	343	375	407	439	471	503	119	w
X	88	280	312	344	376	408	440	472	504	120	x
Y	89	281	313	345	377	409	441	473	505	121	y
Z	90	282	314	346	378	410	442	474	506	122	z
[	91	283	315	347	379	411	443	475	507	123	{
\	92	284	316	348	380	412	444	476	508	124	¦
]	93	285	317	349	381	413	445	477	509	125	}
∧	94	286	318	350	382	414	446	478	510	126	~
_	95	287	319	351	383	415	447	479	511	127	RUBOUT (DEL)
DEC. ⟶		40	41	42	43	44	45	46	47		
ASCII ⟶		(	)	*	+	,	—	.	/		
		High Order X & Y									

Table C-1 (cont)

4010—STYLE GRAPHICS CODE CHART

Low Order X										Low Order Y	
ASCII	DEC.	X or Y Coordinate								ASCII	DEC.
@	64	512	544	576	608	640	672	704	736	`	96
A	65	513	545	577	609	641	673	705	737	a	97
B	66	514	546	578	610	642	674	706	738	b	98
C	67	515	547	579	611	643	675	707	739	c	99
D	68	516	548	580	612	644	676	708	740	d	100
E	69	517	649	581	613	645	677	709	741	e	101
F	70	518	550	582	614	646	678	710	742	f	102
G	71	519	551	583	615	647	679	711	743	g	103
H	72	520	552	584	616	648	680	712	744	h	104
I	73	521	553	585	617	649	681	713	745	i	105
J	74	522	554	586	618	650	682	714	746	j	106
K	75	523	555	587	619	651	683	715	747	k	107
L	76	524	556	588	620	652	684	716	748	l	108
M	77	525	557	589	621	653	685	717	749	m	109
N	78	526	558	590	622	654	686	718	750	n	110
O	79	527	559	591	623	655	687	719	751	o	111
P	80	528	560	592	624	656	688	720	752	p	112
Q	81	529	561	593	625	657	689	721	753	q	113
R	82	530	562	594	626	658	690	722	754	r	114
S	83	531	563	595	627	659	691	723	755	s	115
T	84	532	564	596	628	660	692	724	756	t	116
U	85	533	565	597	629	661	693	725	757	u	117
V	86	534	566	598	630	662	694	726	758	v	118
W	87	535	567	599	631	663	695	727	759	w	119
X	88	536	568	600	632	664	696	728	760	x	120
Y	89	537	569	601	633	665	697	729	761	y	121
Z	90	538	570	602	634	666	698	730	762	z	122
[	91	539	571	603	635	667	699	731	763	{	123
\	92	540	572	604	636	668	700	732	764	¦	124
]	93	541	573	605	637	669	701	733	765	}	125
^	94	542	574	606	638	670	702	734	766	~	126
_	95	543	575	607	639	671	703	735	767	RUBOUT (DEL)	127
DEC ——→		48	49	50	51	52	53	54	55		
ASCII ——→		0	1	2	3	4	5	6	7		
		High Order X & Y									

Table C-1 (cont)

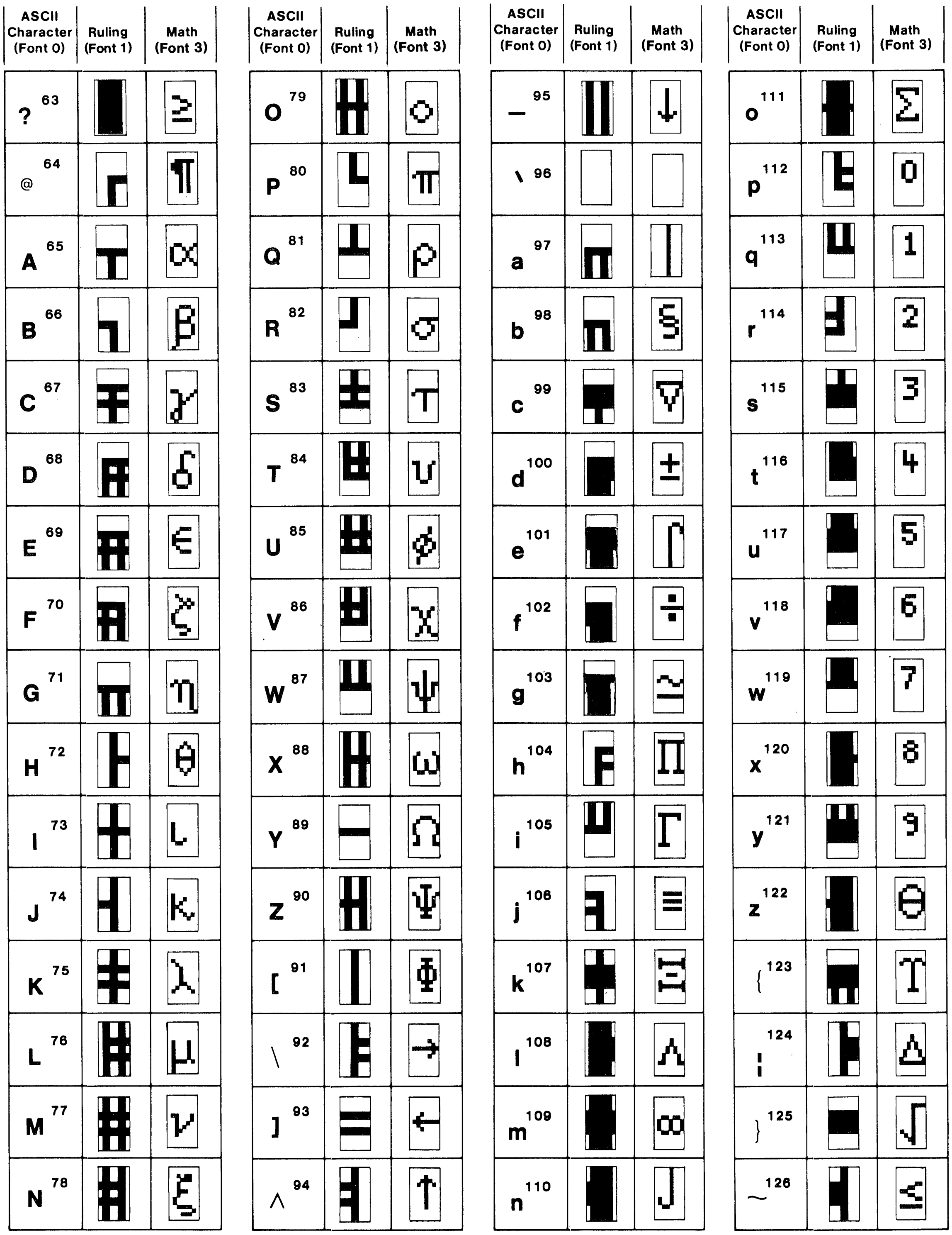
4010—STYLE GRAPHICS CODE CHART

Low Order X		X or Y Coordinate								Low Order Y	
ASCII	DEC.									DEC.	ASCII
@	64	768	800	832	864	896	928	960	992	96	`
A	65	769	801	833	865	897	929	961	993	97	a
B	66	770	802	834	866	898	930	962	994	98	b
C	67	771	803	835	867	899	931	963	995	99	c
D	68	772	804	836	868	900	932	964	996	100	d
E	69	773	805	837	869	901	933	965	997	101	e
F	70	774	806	838	870	902	934	966	998	102	f
G	71	775	807	839	871	903	935	967	999	103	g
H	72	776	808	840	872	904	936	968	1000	104	h
I	73	777	809	841	873	905	937	969	1001	105	i
J	74	778	810	842	874	906	938	970	1002	106	j
K	75	779	811	843	875	907	939	971	1003	107	k
L	76	780	812	844	876	908	940	972	1004	108	l
M	77	781	813	845	877	909	941	973	1005	109	m
N	78	782	814	846	878	910	942	974	1006	110	n
O	79	783	815	847	879	911	943	975	1007	111	o
P	80	784	816	848	880	912	944	976	1008	112	p
Q	81	785	817	849	881	913	945	977	1009	113	q
R	82	786	818	850	882	914	946	978	1010	114	r
S	83	787	819	851	883	915	947	979	1011	115	s
T	84	788	820	852	884	916	948	980	1012	116	t
U	85	789	821	853	885	917	949	981	1013	117	u
V	86	790	822	854	886	918	950	982	1014	118	v
W	87	791	823	855	887	919	951	983	1015	119	w
X	88	792	824	856	888	920	952	984	1016	120	x
Y	89	793	825	857	889	921	953	985	1017	121	y
Z	90	794	826	858	890	922	954	986	1018	122	z
[	91	795	827	859	891	923	955	987	1019	123	{
\	92	796	828	860	892	924	956	988	1020	124	¦
]	93	797	829	861	893	825	957	989	1021	125	}
∧	94	798	830	862	894	926	958	990	1022	126	~
—	95	799	831	863	895	927	959	991	1023	127	RUBOUT (DEL)
DEC →		56	57	58	59	60	61	62	63		
ASCII →		8	9	:	;	<	=	>	?		
		High Order X & Y									

Appendix D

ALTERNATE CHARACTER FONTS

Table D-1
ALTERNATE CHARACTER FONTS



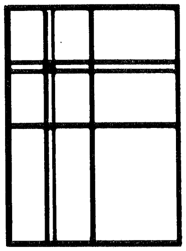
ASCII Character (Font 0)	Ruling (Font 1)	Math (Font 3)	ASCII Character (Font 0)	Ruling (Font 1)	Math (Font 3)	ASCII Character (Font 0)	Ruling (Font 1)	Math (Font 3)	ASCII Character (Font 0)	Ruling (Font 1)	Math (Font 3)
? 63	█	≥	O 79	╫	ο	_ 95	║	↓	o 111	█	Σ
@ 64	┏	¶	P 80	┗	π	\ 96			p 112	┣	0
A 65	┳	∝	Q 81	┻	ρ	a 97	╥	|	q 113	╨	1
B 66	┓	β	R 82	┛	σ	b 98	╥	§	r 114	┫	2
C 67	╪	γ	S 83	╪	τ	c 99	▄	∇	s 115	█	3
D 68	╥	δ	T 84	╫	υ	d 100	█	±	t 116	█	4
E 69	╦	∈	U 85	╪	φ	e 101	█	⌠	u 117	█	5
F 70	╥	ζ	V 86	╫	χ	f 102	█	÷	v 118	▀	6
G 71	╥	η	W 87	╨	ψ	g 103	█	≃	w 119	▀	7
H 72	┣	θ	X 88	╫	ω	h 104	┣	Π	x 120	█	8
I 73	╋	ι	Y 89	━	Ω	i 105	╨	Γ	y 121	▀	9
J 74	┫	κ	Z 90	╢	Ψ	j 106	┫	≡	z 122	█	Θ
K 75	╪	λ	[91	┃	Φ	k 107	╋	Ξ	{ 123	▄	Τ
L 76	╫	μ	\ 92	┣	→	l 108	█	Λ	¦ 124	┣	Δ
M 77	╬	ν	] 93	═	←	m 109	█	∞	} 125	━	√
N 78	╫	ξ	∧ 94	┫	↑	n 110	█	⌡	~ 126	┫	≤

(2402) 4172-5A

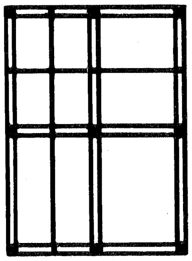
Table D-2

RULING JUNCTIONS CHART

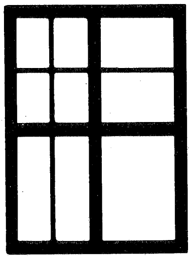
Rulings (Font 1) | **Standard (Font 0)**



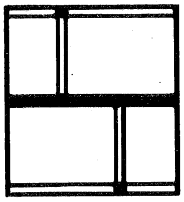
```
@YYGYYAYYYYYB
[  _  [     [
\]]M]]K]]]]]^
[  _  [     [
HYYOYYIYYYYYJ
[  _  [     [
[  _  [     [
[  _  [     [
PYYWYYQYYYYYR
```



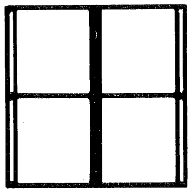
```
D]]C]]E]]]]]F
_  [  _     _
XYYIYYOYYYYYZ
_  [  _     _
L]]K]]M]]]]]N
_  [  _     _
_  [  _     _
_  [  _     _
T]]S]]U]]]]]V
```



```
d))c))e)))))f
?  [  ?     ?
xYYIYYoYYYYYz
?  [  ?     ?
l))k))m)))))n
?  [  ?     ?
?  [  ?     ?
?  [  ?     ?
t))s))u)))))v
```



```
h]]]E]]]]]]]j
[   _       [
[   _       [
:)))y)))()))~
[       _   [
[       _   [
p]]]]]]]U]]]r
```



```
a]]]]]g]]]]]b
_     ?     _
_     ?     _
XYYYYYoYYYYYX
_     ?     _
_     ?     _
qYYYYYwYYYYYi
```

Appendix E

SAMPLE PROGRAMS

This appendix contains two sample programs for the terminal. The first is a short program (actually a segment of a larger program) in PASCAL which processes the input to the host from a terminal !REP00 command. The second is a complete COBOL program which displays a form in the workspace and stores the data from the form in a file.

THE PASCAL PROGRAM

The following PASCAL program issues a !REP00; command to the terminal, analyzes the ANSwer to the host, and returns the terminal status indicated by ANSwer. (See the REPORT command discussion in the Host Programming for the terminal section.)

```
{$t-,d-}
{*** REPORT ***

        This will read the data returned from the
various REPort commands }

VAR     No_more_to_do   : boolean;

PROCEDURE Convert_number(VAR number:integer);
        { This will convert the ASCII character string being input
          by the termianl to INTEGER format.  Any non-numeric character
          terminates the conversion process.  The default value is 0 }

BEGIN
    number := 0;                { set up default value }
    WHILE tty^ in ['0'..'9'] DO BEGIN
        number := number*10 + (ord(tty^)-48);
        get(tty)
        END; {of while}
END; { converting a number }


PROCEDURE Report_1;
        { This will inquire about the system itself }

VAR     Blocks
        ,Bytes
        ,i
        ,Status
                : integer;
```

```
BEGIN
     Write (tty,'!rep 00;');               { Issue report request }
    Break; Reset(tty);
    WHILE tty^ # '!' DO get(tty);          { Search for command character }
    FOR i := 1 to 6 DO get(tty);           { Skip over to the type of ans }
    IF tty^ # '0' THEN Writeln (tty,'<< ANS not of proper type >>')
        ELSE BEGIN
            get(tty); get(tty);                    { skip commas }
            convert_number(blocks);        { get blocks free }
            write (tty,blocks:4,'=Blocks/');
            bytes := blocks*16;            { convert blocks to bytes }
            write (tty,bytes:5,'=Bytes Available');
            get(tty);                      { skip comma }
            convert_number(status);
            CASE status OF
                1 : Write (tty,' < Buffered >');
                2 : Write (tty,' < Form Fill out >');
                3 : Write (tty,' < Buffered & Form Fill out >');
                4 : Write (tty,' < Monitor Present >');
                5 : Write (tty,' < Buffered & Monitor Present >');
                6 : Write (tty,' < Form Fill out & Monitor Present >');
                7 : Write (tty,' < Monitor, Form, & Buffered >')
            END; { of CASE }
    END {If then ELSE}
END; {report 1}

{              * * * *   M A I N   * * * *                          }

BEGIN

Rewrite (ttyoutput);                       { Open tty communications }
No_more_to_do := true;                     { initz }

REPEAT
    Report_1;
    no_more_to_do := true
UNTIL no_more_to_do

END.
```

THE COBOL PROGRAM

The program which follows is written in COBOL and demonstrates the use of form fillout and buffering. The program could have been written in any language which supports some way of writing to and reading from the display character type information.

Design Objectives

Because COBOL works best with defined fields of fixed length and because the 1968 COBOL standard has no string handling verbs, this program was set up to process a field at a time instead of a line at a time.

To make the terminal do this, buffered mode is used, and the FIELD SEPARATOR character is set to carriage return. The EOL string becomes the field separator as well as the normal end-of-line string. In this manner, the ACCEPT always gets either one field followed by an EOL or just an EOL. With SEND MOD, each field of variable form data is preceded by seven characters of row and column information. The amount of form data is variable because trailing blanks are not sent.

This field is ACCEPTED into TERMINAL-DATA which is PICTURed large enough to contain the largest data field from the form plus seven more characters.

Analysing the Input

The program examines TD-ROW. Once the row is identified, a routine is PERFORMed to examine TD-COLUMN. Once the column is identified, a routine is PERFORMed to analyse the data. In this program the data analysis simply stores the data in a file. Because data is being moved from a larger to smaller field, the compiler will generate warning messages telling you of this fact.

End of Data From Terminal

Since this form does not require all of its fields to be entered and since the information is received and processed a field at a time, line 23 was set up as a protected modified field. Therefore, this field is always sent to the host and is sent last. When this field is received, the end of the screen has been reached and everything on that screen has been processed. An appropriate indicator is set to indicate the end of the buffer.

Exiting the Program

To provide an orderly exit from the program, the keyword QUIT is entered into the first field of a new screen. The column processor for that line detects QUIT and sets an end of job indicator which signals the end to the rest of the program. An exit is made after the terminal is returned to a reference state for the next job.

Synchronous Protocol

The following program was run using asynchronous protocol.

```
000100  IDENTIFICATION DIVISION.
000020   PROGRAM-ID. FORM.
000300  REMARKS.
000400*
000500**                THIS IS A SAMPLE PROGRAM WHICH SENDS A FORM
000600**                TO THE 4025 AND PROCESSES THE DATA RETURNED.
000700**
000800** << INPUT/OUTPUT ASSUMPTIONS >>
000900
001000**        THIS PROGRAM USES THE 'ACCEPT' AND 'DISPLAY' VERBS TO
001100** COMMUNICATE WITH THE 4025 VIDEO TERMINAL.  FILE OUTPUT
001200** IS AS PER USUAL.
001300**
001400**        THE INPUT FROM THE 4025 IS VARIABLE IN LENGTH WITH
001500** THE FIRST 7 CHARACTERS ALWAYS PRESENT AND IN THE SAME
001600** FORMAT.
001700**
001800**        FROM 4025:   ROW,COL......DATA......
001900**                      -   - -              -
002000**                      !   ! !              !__ END OF DATA
002100**                      !   ! !_________________ START OF DATA
002200**                      !   !___________________ COLUMN DATA STARTS IN
002300**                      !_______________________ ROW DATA STARTS IN
002400**
002500**        THIS PROGRAM WAS ORIGINALLY DEVELOPED ON A DECSYSTEM-10
002600** AND WITH MODIFICATIONS TO THE 'SELECT' AND 'FD' STATEMENTS AND
002700** CHANGING THE 'DISPLAY-7' TO WHATEVER DISPLAY ALLOWS INPUT OF
002800** OF UPPER AND LOWER CASE CHARACTERS SHOULD CONVERT IT TO
002900** ANOTHER SYSTEM.
005200  ENVIRONMENT DIVISION.
005300  INPUT-OUTPUT SECTION.
005400  FILE-CONTROL.
005500
005600           SELECT   TEK-FORM
005700                    ASSIGN TO DSK
005800                    ACCESS MODE IS SEQUENTIAL
005900                    PROCESSING MODE IS SEQUENTIAL.
006000
```

```
006500
006600  DATA DIVISION.
006700
006800
006900
007000  FILE SECTION.
007200
007300  FD TEK-FORM
007400           VALUE OF ID IS 'DATA  SEQ'
007500           DATA RECORD IS TEK-FORM-DATA.
007600  01 TEK-FORM-DATA                        USAGE IS DISPLAY-7.
007700           02 TF-REF                           PIC X(10).
007800           02 TF-CUST.
007900                   03 TF-CUST-1                PIC X(4).
008000                   03 TF-CUST-2                PIC X(10).
008100           02 TF-INSTRUMENT                    PIC X(8).
008200           02 TF-VALUE                         PIC X(8).
008300           02 TF-TYPE                          PIC X(5).
008400           02 TF-FC-AMOUNT                     PIC X(13).
008500           02 TF-RATE                          PIC X(12).
008600           02 TF-DOLLARS                       PIC X(15).
008700           02 TF-OUR-ACCT                      PIC X(4).
008800           02 TF-CONTRACT-NR                   PIC X(8).
008900           02 TF-CREDIT-REFERENCE              PIC X(12).
009000           02 TF-CHECK-OK                      PIC X(1).
011000
011900 WORKING-STORAGE SECTION.
012000
012200
012300 01 TEK-FORM-FILL-OUT          USAGE IS DISPLAY-7.
012400*=======================
012500
012600      02 TFFO-0                         PIC X(33) VALUE IS
012700      '!FOR N;!BUF;!WOR 24 K H;!JUM;!ERA'.
012800      02 TFFO-1                         PIC X(12) VALUE IS
012900      '!JUM;!ATT IP'.
013000
013100      02 TFFO-1A                        PIC X(19) VALUE IS
013200      '!JUM 1,11;T  E  K   '.
013300
013400      02 TFFO-1B                        PIC X(33) VALUE IS
013500      '!JUM 1,27;SAMPLE OF FORM FILL OUT'.
013600
013700      02 TFFO-1C                        PIC X(27) VALUE IS
013800      '!JUM 1,79; !ATT PS;!JUM 3,1'.
013900*
014000      02 TFFO-3                         PIC X(19) VALUE IS
014100      '!ATT P;REF !JUM 3,6'.
014200
014300      02 TFFO-3A                        PIC X(27) VALUE IS
014400      '!ATT EA;          !JUM 3,16'.
014500
014600      02 TFFO-3B                        PIC X(36) VALUE IS
014700      '!ATT PS;    CUST !ATT EA;    !ATT PS'.
014800
014900      02 TFFO-3C                        PIC X(27) VALUE IS
015000      ' !ATT EA;!JUM 3,39; !ATT PS'.
015100
015200      02 TFFO-3D                        PIC X(46) VALUE IS
015300      '  INSTRUMENT !ATT EA;        !ATT PS;!JUM 3,61'.
015400
015500      02 TFFO-3E                        PIC X(33) VALUE IS
015600      ' VALUE DT !ATT EA;        !ATT PS'.
```

```
015700
015800*
015900*LINE 5
016100*
016200      02 TFFO-5                             PIC X(55) VALUE IS
016300      '!JUM 5;!ATT PS;    TYPE  !ATT EA;      !JUM 5,17;!ATT PS'.
016400
016500      02 TFFO-5A                            PIC X(45) VALUE IS
016600      ' F/C AMT !JUM 5,27;!ATT AE;!JUM 5,40; !ATT PS'.
016700
016800      02 TFFO-5B                            PIC X(33) VALUE IS
016900      '  RATE !ATT EA;!JUM 5,60; !ATT PS'.
017000
017100      02 TFFO-5C                            PIC X(30) VALUE IS
017200      '    $ !ATT FN!JUM 5,79; !ATT PS'.
017300
018000*
018100* LINE 7
018200*

018400*
018500
018600      02 TFFO-7                             PIC X(45) VALUE IS
018700      '!JUM 7;!ATT PS;  OUR ACCT !ATT AE;    !ATT PS'.
018800
018900      02 TFFO-7A                            PIC X(30) VALUE IS
019000      ' CONT# !ATT AE;         !ATT PS'.
019100
019200      02 TFFO-7B                            PIC X(46) VALUE IS
019300      '  CREDIT REFERENCE# !ATT AE;!JUM 7,62; !ATT PS'.
019400
019500      02 TFFO-7C                            PIC X(29) VALUE IS
019600      '    CHECK OK !ATT AE; !ATT PS'.
019700
030100*
030200* LINE 23: SPECIAL END OF SCREEN DATA FLAG
030300*
030400      02 TFFO-EOD-FLAG                      PIC X(45) VALUE IS
030500      '!JUM 23;!ATT PM;01'.
030600*
030700* FORM END
030800*
030900      02 TFFO-END                           PIC X(13) VALUE IS
031000      '!JUM 3,6;!FOR'.
031100
031200 01 TERMINAL-DATA                     USAGE IS DISPLAY-7.
031300*================
031400
031500      02 TD-LOCATION.
031600          03 TD-LOC-ROW                     PIC X(3).
031700          03 FILLER                         PIC X.
031800          03 TD-LOC-COL                     PIC X(3).
031900      02 TD-DATA                            PIC X(40).
032000
032100 01 INDICATORS.
032200*==============
032300
032400      02 END-OF-JOB-IND                     PIC X(3).
032500          88 END-OF-JOB                     VALUE IS 'EOJ'.
032600
032700      02 LINE-IS-EMPTY-IND                  PIC X(3).
032800          88 LINE-IS-EMPTY                  VALUE IS 'EOL'.
032900
```

```
033000 01 CONSTANTS.
033100*=============
033200
033300     02 PROMPT-4025                   PIC X VALUE IS '?'.
033400
033500 01 VARIABLES.
033600*=============
033700
033800     02 TIMES-SCREEN-RESET           PIC 9(2).
034090*                                                   *
034100   PROCEDURE DIVISION.
034200*                                                           *
034400
034600   INITIALIZATION.
034800
034900           OPEN OUTPUT TEK-FORM.
035000           MOVE SPACES TO TEK-FORM-DATA.
035100           PERFORM TERMINAL-SET-UP.
035200           PERFORM DISPLAY-THE-FORM.
035400
035500
035700   MAIN-PART.
035900
036000           MOVE SPACES TO END-OF-JOB-IND.
036100           PERFORM PROCESS-THE-TERMINAL-DATA UNTIL END-OF-JOB.
036200* FLUSH THE LINE 23 FIELD STILL BUFFERED UP
036300           DISPLAY PROMPT-4025.
036310           ACCEPT TERMINAL-DATA.
036320           DISPLAY PROMPT-4025.
036330           ACCEPT TERMINAL-DATA.
036400           CLOSE TEK-FORM.
036500           PERFORM TERMINAL-SET-DOWN.
036600           STOP RUN.
036700* << LOGICAL END OF PROGRAM >>
037000   PROCESS-THE-TERMINAL-DATA.
037200*  -------------------------
037200
037300           MOVE SPACES TO TERMINAL-DATA.
037400           PERFORM GET-BUFFER-LINE.
037500
037600           MOVE SPACES TO LINE-IS-EMPTY-IND
037700           PERFORM DISASSEMBLE-4025-INPUT UNTIL
037800                           LINE-IS-EMPTY
037900           IF NOT END-OF-JOB
038000                   WRITE TEK-FORM-DATA
038100                   DISPLAY PROMPT-4025
038200                   MOVE SPACES TO TEK-FORM-DATA
038300                   DISPLAY PROMPT-4025
038400           ELSE
038500                   NEXT SENTENCE.
038600           DISPLAY '!ERA;!BEL'.
038700*
038800   TERMINAL-SET-UP.
039010*  ---------------
039000
039120* IT IS BEST TO SET THE 4025 TO A KNOWN STATE
039130* RATHER THAN TO ASSUME WHAT STATE IT IS IN.
039100* SET PROMPT := <?><CARRIAGE-RETURN><LINE-FEED>
039200* SET F1 := <CARRIAGE-RETURN><LINE-FEED>
039300* SET PT := <SEND MODIFIED>
039400* SET FIELD-SEPERATOR := <CARRIAGE-RETURN>
039500
039600           DISPLAY '!PRO 63,13,10'.
```

```
039700            DISPLAY '!LEA F1 13,10'.
039800            DISPLAY '!LEA PT /!REP 1;!SEN MOD/13'.
039900            DISPLAY '!FIE 13'.
040000
040200   TERMINAL-SET-DOWN.
040210*  -----------------
040400
040500* PURPOSE:
040600*
040700*       RETURN THE 4025 TO COMMUNICATION WITH THE HOST,
040800* THE KEYBOARD TO THE MONITOR SPACE, NON-FORM FILL OUT,
040900* UNBUFFERED, ALL FUNCTION AND OTHER KEYS UN-LEARNED, AND
041000* DISPLAY IN THE MONITOR SPACE OF THE 4025 A MESSAGE SAYING
041100* WHAT HAS BEEN DONE.

041210*     IN THIS WAY, IT IS POSSIBLE FOR THE USER TO COMMUNICATE
041220* WITH THE COMPUTER WITHOUT LOOSING THE LAST SCREEN.
041200
041300            DISPLAY '!FOR N;!BUF N;!MON K H'.
041400            DISPLAY '!CLEAR'.
041500            DISPLAY '<< END OF TEK FORM FILLOUT EXAMPLE >>'.
041600
041800   GET-BUFFER-LINE.
041810*  ---------------
042000*     SINCE INFORMATION IS 'BUFFERED' , IT IS NECESSARY TO
042010* 'PROMPT' THE 4025 TO SEND THE DATA AND THEN WAIT FOR IT TO
042020* BE SENT.
042300
042400            DISPLAY PROMPT-4025.
042500            ACCEPT TERMINAL-DATA.
042510
043000   DISASSEMBLE-4025-INPUT.
043010*  ----------------------
043200
043300*      THIS SECTION IS A 'CASE' STATEMENT THT EXAMINES
043400* THE DATA FROM THE 4025 AND PERFORMS A ROUTINE
043500* WHICH FURTHER EXAMINES THE LINE.
043600*
043700*  CASE TD-LOC-ROW OF
043800*       '003' : PERFORM ROW 3 PROCESSING
043900*       '005' : PERFORM ROW 5 PROCESSING
044100*       '023' : PERFORM ROW 23 PROCESSING
044300*
044400* WHEN THE PROTECTED MODIFIED FIELD ON LINE 23 IS PROCESSED, EOL IS
044500* SET TRUE AND THE BUFFER CONSIDERED PROCESSED.  THE WHOLE PROCESS
044600* STARTS OVER AGAIN AND A NEW BUFFER IS PROCESSED.
044700*
044800            IF TD-LOC-ROW = '003'
044900                    PERFORM ROW-3
045000
045100            ELSE IF TD-LOC-ROW = '005'
045200                    PERFORM ROW-5
045210
045220            ELSE IF TD-LOC-ROW = '023'
045230                    PERFORM ROW-23.
045300
046800* NOTE:    IF ADDITIONAL LINES EXIST ON THE FORM
046900*          ADDITIONSL 'ELSE IF' LINES MAY BE CODED.
047000* HOWEVER, THERE IS A LIMIT TO THE DEPTH TO WHICH
047100* MOST COMPILERS WILL ALLOW IF'S TO BE NESTED.
047200*          TO OVERCOME THE NESTING LIMIT, START UP AN-
047300* OTHER 'IF...THEN...ELSE'.
```

```

           IF LINE-IS-EMPTY
                   NEXT SENTENCE
           ELSE   PERFORM GET-BUFFER-LINE.

*
* WORK ROUTINES
*
*      THESE ROUTINES FURTHER BREAK DOWN THE DATA JUST RECEIVED
* FROM THE 4025 TERMINAL.  EACH OF THE ROUTINES THAT FOLLOW
* IS DEDICATED TO ONE ROW FROM THE 4025.  THESE ROUTINES WILL
* EXAMINE THE COLUMN NUMBER AND TAKE APPROPRIATE ACTION.
* IN THIS PROGRAM THE DATA IS JUST SAVED IN A DISK FILE

* BUT ADDITIONAL PROCESSING COULD HAVE BEEN DONE.
*

*******
 ROW-3.
*******

           IF TD-LOC-COL = '006'
                   MOVE TD-DATA TO TF-REF

           ELSE IF TD-LOC-COL = '025'
                   MOVE TD-DATA TO TF-CUST-1

           ELSE IF TD-LOC-COL = '030'
                   MOVE TD-DATA TO TF-CUST-2

           ELSE IF TD-LOC-COL = '053'
                   MOVE TD-DATA TO TF-INSTRUMENT

           ELSE IF TD-LOC-COL = '071'
                   MOVE TD-DATA TO TF-VALUE

           ELSE
                   NEXT SENTENCE.

           IF TF-REF = 'QUIT'
                   MOVE 'EOL' TO LINE-IS-EMPTY-IND
                   MOVE 'EOJ' TO END-OF-JOB-IND
           ELSE
                   NEXT SENTENCE.

*******
 ROW-5.
*******

           IF TD-LOC-COL = '011'
                   MOVE TD-DATA TO TF-TYPE

           ELSE IF TD-LOC-COL = '027'
                   MOVE TD-DATA TO TF-FC-AMOUNT

           ELSE IF TD-LOC-COL = '048'
                   MOVE TD-DATA TO TF-RATE

           ELSE IF TD-LOC-COL = '066'
                   MOVE TD-DATA TO TF-DOLLARS

           ELSE
```

```
056500                         NEXT SENTENCE.
056600
056700*******
056800 ROW-7.
056900*******
057100              IF TD-LOC-COL = '012'
057200                         MOVE TD-DATA TO TF-OUR-ACCT
057300
057400              ELSE IF TD-LOC-COL = '023'
057500                         MOVE TD-DATA TO TF-CONTRACT-NR
057600
057700              ELSE IF TD-LOC-COL = '051'
057800                         MOVE TD-DATA TO TF-CREDIT-REFERENCE
057900
058000              ELSE IF TD-LOC-COL = '076'
058100                         MOVE TD-DATA TO TF-CHECK-OK
058200
058300              ELSE
058400                         NEXT SENTENCE.
058500
068100
068200********
068300 ROW-23.
068400********
068500
068600              MOVE 'EOL' TO LINE-IS-EMPTY-IND.
068700
068800   DISPLAY-THE-FORM.
071300*  ----------------
071400
071500              DISPLAY TFFO-0.
071600              DISPLAY TFFO-1.
071700              DISPLAY TFFO-1A.
071800              DISPLAY TFFO-1B.
071900              DISPLAY TFFO-1C.
072000              DISPLAY TFFO-3.
072100              DISPLAY TFFO-3A.
072200              DISPLAY TFFO-3B.
072300              DISPLAY TFFO-3C.
072400              DISPLAY TFFO-3D.
072500              DISPLAY TFFO-3E.
072600              DISPLAY TFFO-5.
072700              DISPLAY TFFO-5A.
072800              DISPLAY TFFO-5B.
072900              DISPLAY TFFO-5C.
073100              DISPLAY TFFO-7.
073200              DISPLAY TFFO-7A.
073300              DISPLAY TFFO-7B.
073400              DISPLAY TFFO-7C.
075600              DISPLAY TFFO-EOD-FLAG.
075700*
075800*
075900              DISPLAY TFFO-END.
076000*
076100* << PHYSICAL END OF PROGRAM >>
```

Appendix F

COMMAND LISTING

Command	Options Required	Action Different In Form Fillout Mode	Other Comments	Discussed In These Sections
ALLOCATE	4			10
ATTRIBUTE		Logical attributes effective only in Form Fillout Mode	Workspace Only	7
BACKTAB		X		6, 7
BAUD				5
BELL				6
BUFFERED				5
CIRCLE	25 or 26, 38			9
CLEAR				4
COMMAND				2, 4, 5
4025A COPY	3 or 4	X		10
DCHAR		X		8
DELAY				5
DFONT	25 or 26			9
DIRECTORY	4			10
DISABLE	25 or 26, 38			9
DISCONNECT	1			5
DLINE		X		8
DOWN				6
DUPLEX	1			5
ECHO				5
ENABLE	25 or 26, 38			9
EOF	3 or 4			5, 10
EOL				5
ERASE		X		6, 7, 9
EXPAND	25 or 26, 38			4
FIELD				5, 7
FORM				5, 7
GRAPHIC	25 or 26			9
GTEST	25 or 26			5
HCOPY				10
HRULE	32			7
ICHAR		X		8

Command	Options Required	Action Different In Form Fillout Mode	Other Comments	Discussed In These Sections
ILINE		X		8
INK	25 or 26, 38			9
JUMP			Workspace Only	6, 7
KILL	4			10
LEARN				4
LEFT				6
LINE	25 or 26			9
MARGINS			Workspace Only	5
MONITOR				5
PARITY				5
PASS	4			10
PERIPHERALS	4			10
PROMPT				5
REPORT			From the Host only. Some forms require options.	3, 10
RDOWN				6
RIGHT				6
RUP				6
RVECTOR	25 or 26			9
SEND		X		3, 7, 9
SET	3 or 4			10
SHRINK	25 or 26			9
SNOOPY				5
STOPS				5
STRING	25 or 26			9
SYMBOL	25 or 26			9
SYSTAT				5
TAB		X		6, 7
TEST				5
UP				6
VECTOR	25 or 26			9
VRULE	32			7
WORKSPACE				5

Appendix G

PROGRAMMER'S REFERENCE TABLE

If you wish, you may record the following settings for future reference:

Table G-1

PROGRAMMER'S REFERENCE

Transmitting Baud Rate	TB=
Receiving Baud Rate	RB=
Command Character	CC=
Prompt String	PR=
End-of-line String	EL=
Remote Start Stop	RS=
Duplex (Full or Half)	DU=
Echo (Remote or Local)	EC=
Parity (None, Even, Odd, High, or Data)	PA=
Field Separator	FS=
End-of-File String	EF=
Send Key String	
Display Memory Capacity	
Graphics Memory Capacity	
Fonts Installed	
Options Installed	

Appendix H

OPTION SUMMARY

Option 01: Half Duplex

Permits half duplex communications, in addition to the full duplex data communications provided as standard equipment.

Option 02: Current Loop

Permits the terminal to communicate with the host computer or another device by means of a 20 mA current loop rather than the standard RS-232 interface.

Option 03: RS-232 Peripheral Interface (Requires Option 36)

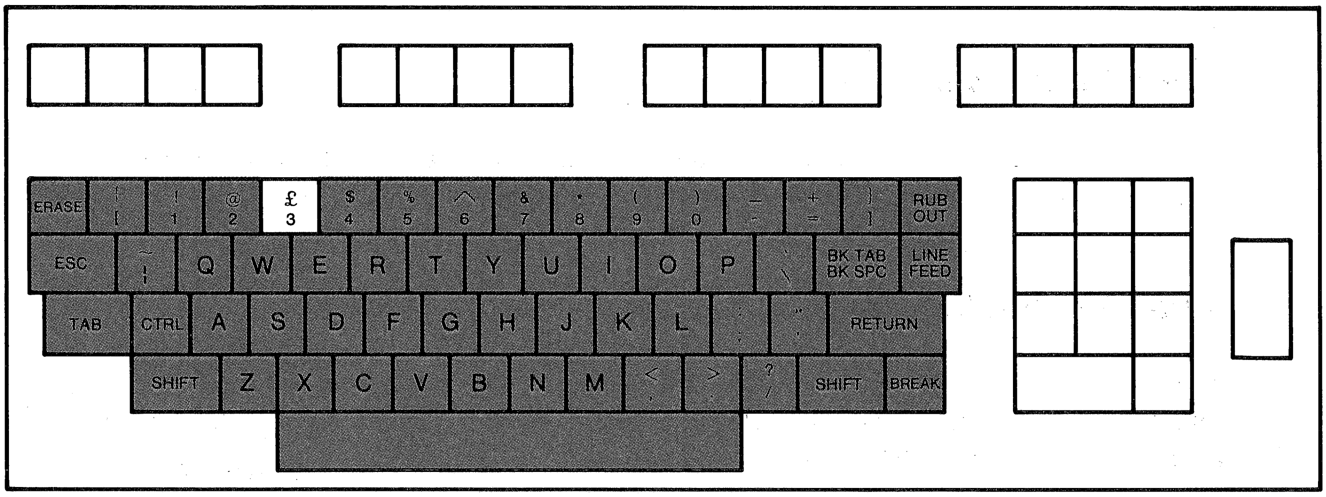
Permits the terminal to transmit to RS-232 compatible peripheral devices such as the TEKTRONIX 4642 Printer. With this option, data from the host computer or the workspace can be printed on the 4642 Printer.

Option 04: GPIB Peripheral Interface (Requires Option 36)

Permits the terminal to communicate with and control the TEKTRONIX 4924 Digital Cartridge Tape Drive and 4662 Interactive Digital Plotter. These devices communicate with the terminal over the General Purpose Interface Bus (GPIB), which is defined in IEEE Standard 488-1975. Allows the terminal to save data or command files on the 4924, and retrieve them later without the need for intervention by the host computer.

Option 4A: United Kingdom Character Set

This option permits Tektronix 4020 Series terminals to change to a United Kingdom standard keyboard layout so that the United Kingdom characters are displayed. The only change is that the "#" sign is replaced by the English "£" sign. This is shown in the revised keyboard configuration (see Figure H-1), and the revised ASCII Code Chart (see Table H-1). When this key is pressed (or the appropriate code is received by the terminal), the "£" sign is displayed on the screen.



2943-1A

Figure H-1. United Kingdom Keyboard.

Table H-1

UNITED KINGDOM CHARACTER SET

B7 B6 B5				Ø Ø Ø	Ø Ø 1	Ø 1 Ø	Ø 1 1	1 Ø Ø	1 Ø 1	1 1 Ø	1 1 1
BITS				CONTROL		HIGH X & Y GRAPHIC INPUT		LOW X		LOW Y	
B4	B3	B2	B1								
Ø	Ø	Ø	Ø	NUL 0	DLE 16	SP 32	0 48	@ 64	P 80	\ 96	p 112
Ø	Ø	Ø	1	SOH 1	DC1 17	! 33	1 49	A 65	Q 81	a 97	q 113
Ø	Ø	1	Ø	STX 2	DC2 18	" 34	2 50	B 66	R 82	b 98	r 114
Ø	Ø	1	1	ETX 3	DC3 19	£ 35	3 51	C 67	S 83	c 99	s 115
Ø	1	Ø	Ø	EOT 4	DC4 20	$ 36	4 52	D 68	T 84	d 100	t 116
Ø	1	Ø	1	ENQ 5	NAK 21	% 37	5 53	E 69	U 85	e 101	u 117
Ø	1	1	Ø	ACK 6	SYN 22	& 38	6 54	F 70	V 86	f 102	v 118
Ø	1	1	1	BEL 7	ETB 23	' 39	7 55	G 71	W 87	g 103	w 119
1	Ø	Ø	Ø	BS 8	CAN 24	(40	8 56	H 72	X 88	h 104	x 120
1	Ø	Ø	1	HT 9	EM 25	) 41	9 57	I 73	Y 89	i 105	y 121
1	Ø	1	Ø	LF 10	SUB 26	* 42	: 58	J 74	Z 90	j 106	z 122
1	Ø	1	1	VT 11	ESC 27	+ 43	; 59	K 75	[91	k 107	{ 123
1	1	Ø	Ø	FF 12	FS 28	, 44	< 60	L 76	\ 92	l 108	¦ 124
1	1	Ø	1	CR 13	GS 29	- 45	= 61	M 77	] 93	m 109	} 125
1	1	1	Ø	SO 14	RS 30	. 46	> 62	N 78	∧ 94	n 110	~ 126
1	1	1	1	SI 15	US 31	/ 47	? 63	O 79	_ 95	o 111	RUBOUT (DEL) 127

2943-2

Option 4B: French Character Set

This option permits Tektronix 4020 terminals to change to the French "AZERTY" keyboard layout for the standard ASCII character set. All the characters are the same as on the 4020 standard keyboard. The only changes are that four keys are switched around. This is shown in the revised keyboard configuration (see Figure H-2). There are no changes to the ASCII Code Chart. When these four keys are pressed (or the appropriate codes are received by the terminal), these characters are displayed on the screen.

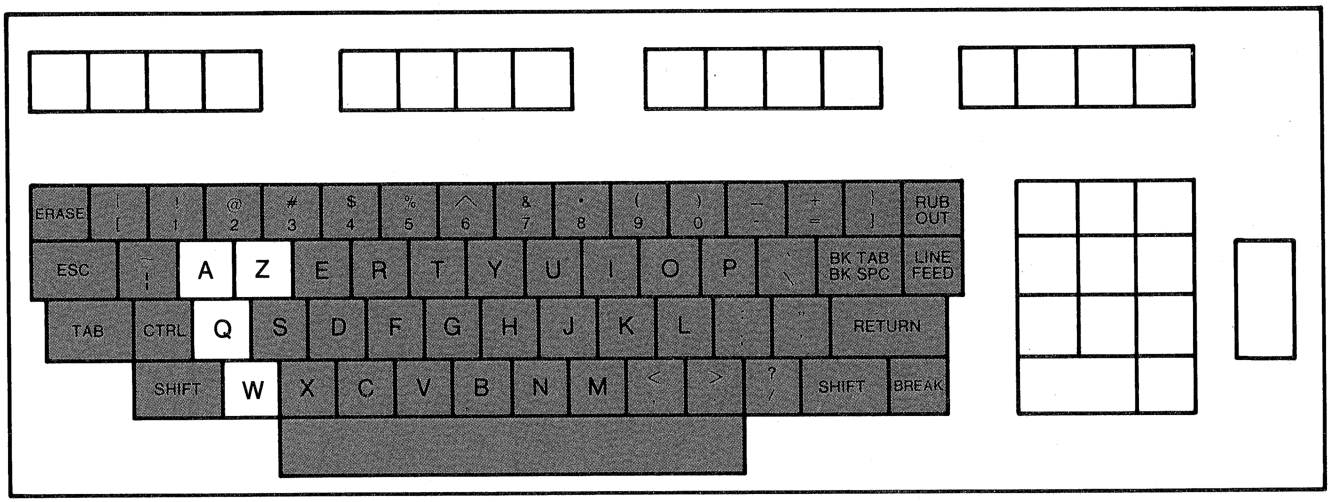


Figure H-2. French Keyboard.

Option 4C: Swedish Character Set

This option permits Tektronix 4020 terminals to change to a Swedish standard layout so that the Swedish characters are displayed. There are 17 changes to the keyboard, with three of these changes being new characters. These changes are shown in the revised keyboard configuration (see Figure H-3), and the revised ASCII Code Chart (see Table H-2). When these 17 keys are pressed (or the appropriate codes are received by the terminal), these characters are displayed on the screen.

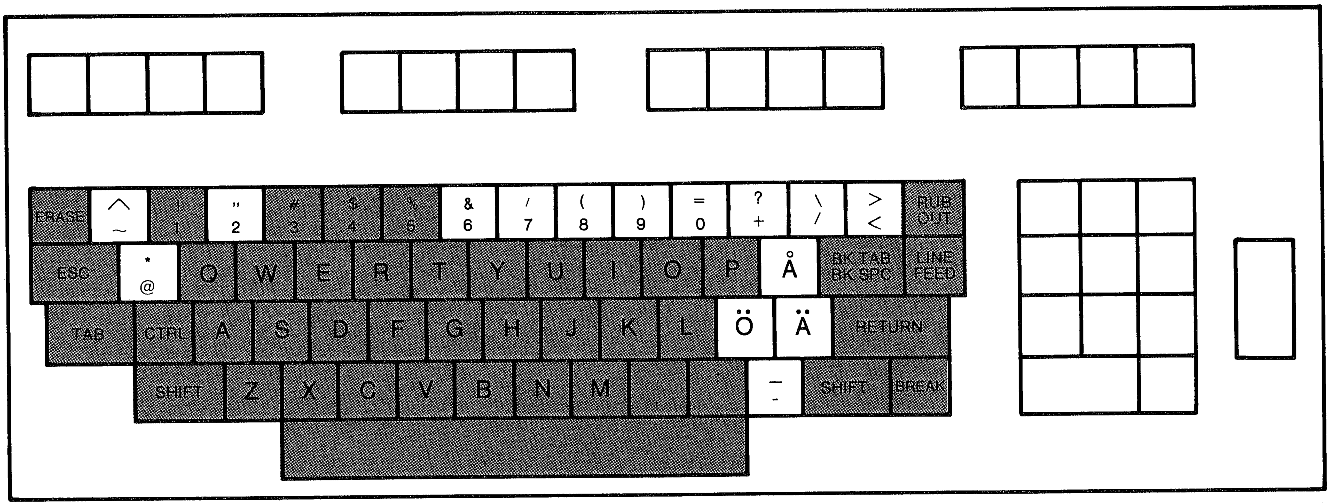


Figure H-3. Swedish Keyboard.

Table H-2

SWEDISH CHARACTER SET

B7 B6 B5 BITS				Ø Ø Ø	Ø Ø 1	Ø 1 Ø	Ø 1 1	1 Ø Ø	1 Ø 1	1 1 Ø	1 1 1
B4	B3	B2	B1	CONTROL		HIGH X & Y GRAPHIC INPUT		LOW X		LOW Y	
Ø	Ø	Ø	Ø	NUL 0	DLE 16	SP 32	0 48	@ 64	P 80	\ 96	p 112
Ø	Ø	Ø	1	SOH 1	DC1 17	! 33	1 49	A 65	Q 81	a 97	q 113
Ø	Ø	1	Ø	STX 2	DC2 18	″ 34	2 50	B 66	R 82	b 98	r 114
Ø	Ø	1	1	ETX 3	DC3 19	# 35	3 51	C 67	S 83	c 99	s 115
Ø	1	Ø	Ø	EOT 4	DC4 20	$ 36	4 52	D 68	T 84	d 100	t 116
Ø	1	Ø	1	ENQ 5	NAK 21	% 37	5 53	E 69	U 85	e 101	u 117
Ø	1	1	Ø	ACK 6	SYN 22	& 38	6 54	F 70	V 86	f 102	v 118
Ø	1	1	1	BEL 7	ETB 23	′ 39	7 55	G 71	W 87	g 103	w 119
1	Ø	Ø	Ø	BS 8	CAN 24	(40	8 56	H 72	X 88	h 104	x 120
1	Ø	Ø	1	HT 9	EM 25	) 41	9 57	I 73	Y 89	i 105	y 121
1	Ø	1	Ø	LF 10	SUB 26	* 42	: 58	J 74	Z 90	j 106	z 122
1	Ø	1	1	VT 11	ESC 27	+ 43	; 59	K 75	Ä 91	k 107	ä 123
1	1	Ø	Ø	FF 12	FS 28	, 44	< 60	L 76	Ö 92	l 108	ö 124
1	1	Ø	1	CR 13	GS 29	– 45	= 61	M 77	Å 93	m 109	å 125
1	1	1	Ø	SO 14	RS 30	. 46	> 62	N 78	^ 94	n 110	~ 126
1	1	1	1	SI 15	US 31	/ 47	? 63	O 79	_ 95	o 111	RUBOUT (DEL) 127

2947-2

Option 22: Added Display Memory

The standard terminal includes 16,384 bytes of display memory. (Each byte is 8 binary bits, and can hold one ASCII character.) Option 22 expands this, permitting larger quantities of text to be stored in the workspace and monitor.

Option 22: A total of 32,768 (32K) bytes of display memory.

Options 25 and 26: Graphics Memory

Permits the terminal to draw graphs in its workspace. Solid lines and seven types of dashed lines can be drawn, and individual points can be plotted. Individual lines can be erased by drawing over them with "erase vectors."

In addition, the Graphics Memory options permit the user to create alternate character fonts for displaying text in the workspace. This is done by using commands to define each character of each new font.

These options differ only in the amount of graphics memory they include. Larger amounts of graphics memory permit the terminal to draw larger and more complex graphs in its workspace, and to create more alternate character sets.

Option 25: 16,384 (16K) bytes of graphics memory.

Option 26: 32,768 (32K) bytes of graphics memory.

Option 31: Character expansion.

Permits the addition of ROMs (Read Only Memories) containing alternate character fonts.

Option 32: Ruling Characters

Adds the ruling character font, permitting single and double lines to be drawn on forms in the workspace, as well as ruling junctions characters.

Option 34: Math Characters (Requires Option 31)

Adds a set of math characters to permit mathematical symbols to be displayed in the workspace. Includes standard mathematical symbols, Greek letters, and superscripts.

Option 36: Peripherals ROM

Provides instructions for the terminal processor, allowing it to communicate with RS-232 or GPIB peripheral devices. (Required for Option 03 or Option 04.)

Option 38: GIN Mode (Requires Options 25 or 26)

Provides for digitized graphics input to the host computer.

Power Cord Options

Option A1	220V/16A 50 Hz operation, universal European plug.
Option A2	240V/13A 50 Hz operation, United Kingdom plug.
Option A1	240V/10A 50 Hz operation, Australian plug.
Option A1	240V/15A 50 Hz operation, North American plug.

Appendix I

BUFFERED MODE PROGRAMMING TIPS

1. When buffering multiple lines to be prompted by the host, enter and exit buffered mode with !BUF and !BUF N commands issued by the host (rather than typed on the keyboard, as by pressing a programmed function key).

   Reason:

   This assures consistent operation. Keyboard-entered commands have more variables involved in execution, such as (a) whether the terminal is already in buffered mode, (b) whether there are lines waiting to be sent in the transmit buffer, and (c) whether there is an outstanding prompt waiting for a line to appear in the transmit buffer.

2. The host software should always, after sending a !BUF command to the terminal, follow that command with a prompt string. (The execution is when the host is using the !BUF to lock the terminal's workspaces, as described below.)

   Reason:

   A prompt is required to send the first line, and to unlock the workspace.

   Workspace. When the keyboard is typing into the workspace, !BUF locks the workspace. (The operator may type ahead, but what he types wil not be entered into the workspace until a prompt is received.) This feature is useful if the host application program calls for the operator to wait.

   Monitor. When the keyboard is typing into the monitor, !BUF with no prompt holds the first line in the transmit buffer until (a) a prompt string is received, and (b) the prompt delay interval has elapsed after the prompt. When both the prompt and the delay have occurred, the line is transmitted.

3. Allow 500 msec after assigning workspace/monitor (!WOR nn or !MON nn command), before proceeding with a buffered command (!BUF) and a prompt string, etc.

   Reason:

   The terminal's processor cannot recognize the prompt string while reassigning the split screen.

4. If you need to cancel an oustanding prompt, have the host send a !BUF command with no prompt.

   Reason:

   The !BUF command, when not followed by a prompt, "disarms" the transmit buffer.

5. When requesting terminal status, cancel any outstanding prompt (!BUF) before sending the request for status (!REP). For example:

   ```
   !BUF!REP00 <CR> prompt
   ```

   Reason:

   This assures that the !ANS reply will go through the delay sequence and attach a CR to the end of the string. The prompt in the example causes the !ANS to transmit. <NOTE 1> This leaves the terminal in buffered mode, waiting for a prompt; so after the host reads the !ANS it should send another prompt.

   <NOTE 1> If the !ANS goes into an empty transmit buffer, or following a line typed in the monitor, it is the next line sent. However, if a !SEND command has copied the workspace into the transmit buffer, the !ANS reply is added at the end of the transmit buffer. The !ANS reply will not be sent to the host until it reaches the head of the transmit buffer queue.

6. The DELAY time should be set to a minimum of 20 msec (!DEL 20), or one character time (11/BAUD) at baud rates below 600. Also, consider the minimum delay which the host software may require before it is ready to receive.

   Reason:

   This assures that the time between characters will not be seen as prompt delay time.

7. After sending a prompt string, the host should always wait (send nothing) for a time delay greater than the !DEL setting.

   Reason:

   The DELAY is not only the time between prompt and transmit, but also the time which the terminal measures to distinguish text from a prompts string.

8. When in buffered mode, do not allow the host to send more than eight prompt strings within one delay time. For example, with DL= 1000 and PR= LF, ten short lines of text (each with LF) could be transmitted from host to terminal well within the one second DELAY. This will hang the terminal's processor.

   Reason:

   The routine which checks prompt strings against delay time can store only nine prompts (eight plus one outstanding).

   The problem can be avoided by using a prompt string not used in text, such as "$< LF>" or "GIMME:". Also, the delay time should be set as short as the host software can handle, considering the minimums stated in item 6 above.

9. The host software should precede any commands with !BUF, to cancel any outstanding prompt. Then follow the commands with prompt when ready to proceed.

   Example:

   !BUF< CR> ... !JUM 16 < CR> ... prompt

   Reason:

   The terminal's processor cannot handle simultaneous commands and communication.

   This method should also be used in unbuffered mode, using !BUF N command to restore the terminal to unbuffered mode when the host is done sending commands.

10. The host software should prompt the terminal buffer to empty it before sending !BUF N, to assure that no lines remain.

    Reason:

    If lines are in the transmit buffer awaiting a sequence of host prompts, the terminal goes into unbuffered mode on !BUF N, but saves the remaining lines. These lines will be sent prior to the next unbuffered transmission from either workspace or monitor, which may cause a problem.

11. To avoid possible problems of prompting an empty buffer (which may cause the host software to hang in READ mode), some sort of "flag" is needed to tell the host when the entire workspace contents has been sent. A convenient way to do this is with the !REPORT command, which causes the terminal to send a !ANS reply.

    Example:

    The host sends "!BUF!SEND!EP00;" to the terminal. This puts the terminal in buffered mode; tells it to send the workspace contents; and tells it to follow those contents with a "!ANS" reply to REPORT command.

    The terminal responds by going into buffered mode, copying workspace contents into its transmit buffer, and copying the !ANS reply into the transmit buffer, after the workspace contents. A separate process in the terminal then responds to host prompts, sending a line from the transmit buffer for each prompt received.

    When the host hears the "!ANS" reply, it knows that it has received the entire workspace contents. At this point, the host may issue " !BUF NO;" to remove the terminal from buffered mode.