TEK HOST PROGRAMMER'S MANUAL

Part No. 061-2616-01
Product Group 16

4113
COMPUTER DISPLAY TERMINAL

Tektronix®
COMMITTED TO EXCELLENCE

Part No. 061-2616-01
Product Group 16

4113
COMPUTER DISPLAY TERMINAL

Please Check for CHANGE INFORMATION at the Rear of this Manual

First Printing DEC 1981
Revised OCT 1982

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.

MANUAL REVISION STATUS

PRODUCT: 4113 Computer Display Terminal

This manual supports the following versions of this product: Serial Numbers B010100 and up.

REV DATE	DESCRIPTION
DEC 1981	Original Issue
AUG 1982	Revised: pages 5-18, 7-11, 7-18, 12-12, 12-19, and C-48.

CONTENTS

Section 3 **THE PROGRAMMING TASK** **Page**

Section 4 **DISPLAY AND KEYBOARD SETTINGS**

Section 5 DISPLAYING GRAPHIC INFORMATION Page

Section 6 **SEGMENTS** **Page**

Section 7 **THE 4113 COLOR DISPLAY** **Page**

Section 8 **GRAPHIC INPUT** **Page**

Section 9 **REQUESTING REPORTS FROM THE TERMINAL** **Page**

Section 10 **COMMUNICATIONS SETTINGS**

Section 11 OPTION 01: HALF DUPLEX AND BLOCK MODE Page

Section 12 PERIPHERAL DATA TRANSFERS **Page**

ILLUSTRATIONS

TABLES

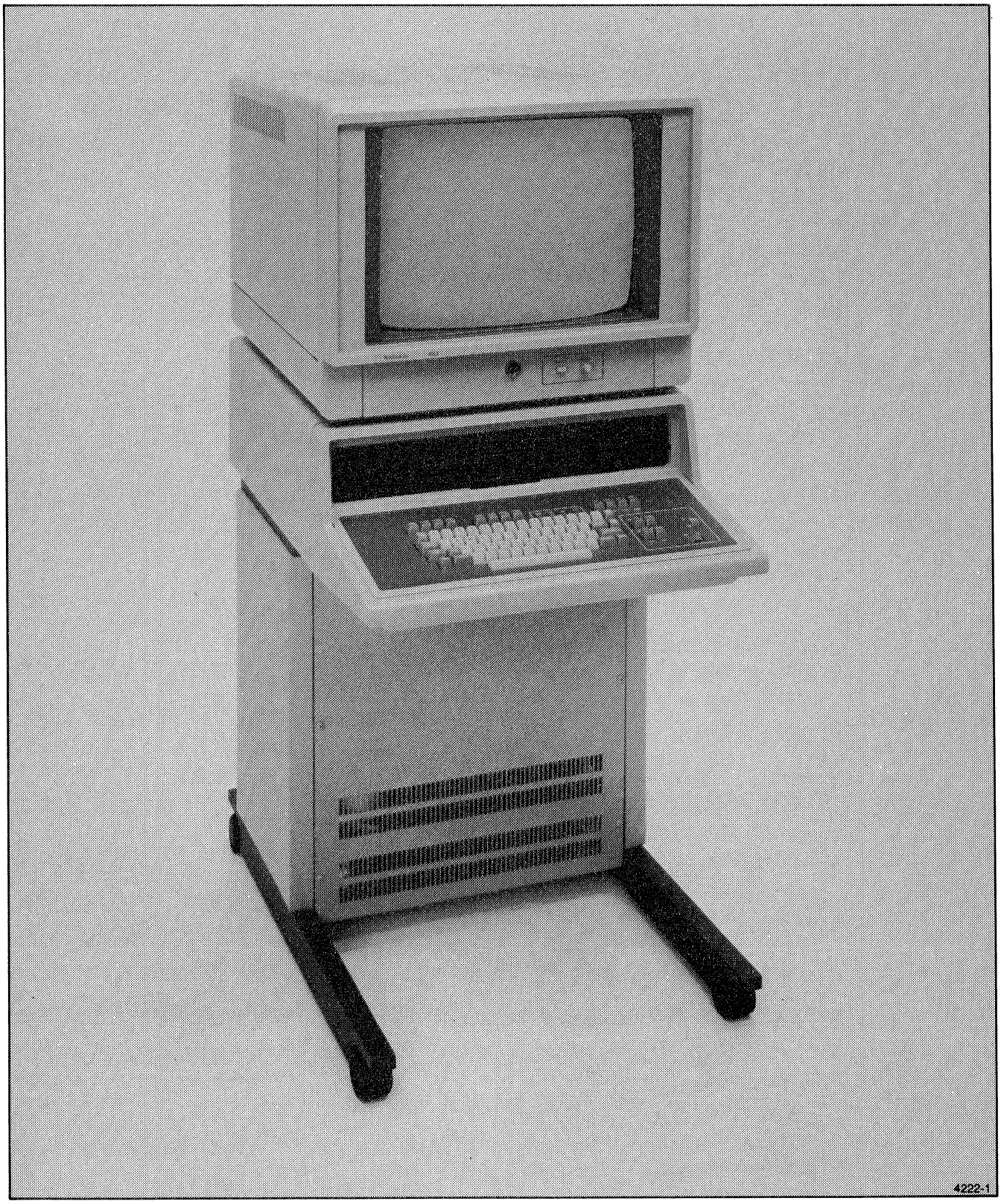


Figure 1-1. 4113 Computer Display Terminal.

Section 1

INTRODUCTION

ABOUT THE MANUAL PACKAGE

THIS MANUAL

This 4113 Host Programmer's Manual is for the programmer who is writing a package of host computer subroutines to communicate with the TEKTRONIX 4113 Computer Display Terminal (Figure 1-1). This manual describes the 4113 commands and tells how to issue them by sending sequences of ASCII characters to the terminal.

NOTE

This manual is intended to be used together with the 4110 Series Command Reference Manual. That manual has, in alphabetical order, descriptions of all the 4113 commands, parameter types, and other syntactic constructs. There you will find the exact syntax of each command, a description of the command's function, a list of error codes for that command, and references to other, related commands.

For operator information (such as the use of SETUP mode), see the 4113 Operator's Manual.

The TEKTRONIX 4113A Computer Display Terminal is a new version of the 4113. The 4113A30 is a desk configuration of the 4113A. All references in this manual to the 4113 apply equally to the 4113A and 4113A30.

OTHER 4113 MANUALS

The following related manuals may prove useful:

- **4113 Computer Display Terminal Operator's Manual.** This manual is written for the terminal operator. Here are descriptions of all the keys and of the "SETUP mode" commands by which the operator can change terminal settings.
- **4110 Series Command Reference Manual.** The Command Reference Manual contains definitive descriptions of all commands, parameter types, report messages, and other syntactic constructs. Once you have become acquainted with the terminal, you will use this manual more than any other.
- **Manuals for the TEKTRONIX 4010C01 PLOT 10 Interactive Graphics Library.** The Interactive Graphics Library (IGL) is a package of FORTRAN subroutines for use in writing computer graphics applications programs. If IGL is available at your computer installation, the easiest way to control the terminal will be through calls to IGL subroutines.

Service manuals are also available; however, these are probably not of use to the host computer programmer.

HOW TO USE THIS MANUAL

BASIC ORGANIZATION

The sections of this manual are arranged in approximate order of increasing complexity. That is, later sections describe more complex subject matter and may employ concepts introduced in earlier sections.

This manual is intended to be read in conjunction with the 4110 Series Command Reference Manual. Be sure to have that manual close at hand; you will need it for descriptions of the exact syntax of the different commands.

Read Section 2 of this manual before reading later sections, and before referring to the 4110 Series Command Reference Manual. Section 2 describes the syntax notation used in the two manuals and describes the general format of 4113 commands.

Section 3 provides an overview of the graphics programming task. If you are already well acquainted with programming techniques, then you may wish to skip this section. Terms such as *device drivers* and *communications interface routines* are defined, and examples of how to write host device driver subprograms are given.

The remaining sections describe different aspects of the terminal, and how to perform common tasks: displaying alphanumeric text, displaying graphic information, using the terminal's local storage of picture segments, etc. If you wish, you may skip to the section describing the task before you. Sooner or later, however, you should read through all these sections to become acquainted with the terminal's most important features.

WHERE TO LOOK FOR INFORMATION

If you are writing a subroutine to issue a single command to the terminal, see the description of that command in the 4110 Series Command Reference Manual. There you will find the command's exact syntax, a description of its function, a list of possible errors which the terminal may detect while executing that command, and so on.

If you a writing a program or subroutine to perform some higher-level function, see the description of that function in this manual. The examples in those sections should help you decide which commands to send to the terminal. You will probably then call subroutines to issue those commands.

ABOUT THE 4113

FEATURES

The 4113 Computer Display Terminal permits the terminal's operator to communicate with a host computer. Besides sending and receiving characters of text, the terminal can:

- Display, in color, pictures sent from the computer or stored locally on its optional disk drive. The colors used for drawing pictures are selected from 4096 possible color mixtures. Of these, as many as eight color mixtures may be used at any one time. (If the terminal is equipped with Option 21, then up to sixteen color mixtures may be used at once.)
- Store and manipulate picture segments locally (within itself). This relieves the host computer of the burden of manipulating those segments. It also makes more efficient use of the data communications line between the terminal and the host computer.
- Cope with imperfect data communications lines by means of a optional block mode communications protocol.
- Create (on command from the host computer) alternate character fonts, and use these fonts to display alphanumeric information.
- Define "macros" which can be expanded on command. Macros numbered from 0 to 143 can also be invoked by pressing keyboard keys; thus the macro facility allows the operator or the host computer to program alternate meanings into most of the terminal's keys.
- Store picture segments and macro definitions on its optional disk drive, and retrieve them later to use again.
- Define "panels" (polygon regions), and fill those panels with predefined fill patterns.
- Define fill patterns for filling panel interiors.

ALPHA, VECTOR, AND MARKER MODES

The 4113 has three main modes of operation: alpha mode, vector mode, and marker mode. There are also several other modes which it enters temporarily while executing certain commands.

- In alpha mode, the 4113 displays characters received from the host. This is the mode used when logging in on a host computer or when running non-graphics programs.
- In vector mode, the 4113 interprets alphanumeric characters coming from the host as <xy> coordinates for "moves" and "draws" of *vectors* (straight line segments) on the screen.
- In marker mode, the 4113 interprets alphanumeric characters coming from the host as <xy> coordinates at which to display *markers* — small symbols.

SEGMENTS

The 4113 can manipulate *segments* (short for "picture segments") and use them to build complex pictures. For instance, segments holding symbols for transistors, resistors, capacitors, etc., might be used to compose an electrical schematic diagram.

The 4113 can store and manipulate these segments locally. This relieves the host computer of some computations. More importantly, the 4113's local segment-handling capability drastically reduces the number of characters which must be sent over the data communications line. (The communications line is the main "bottleneck" inhibiting the rapid display of computer graphics. Graphic coordinates can be transmitted over typical communications lines only at rates far slower than those coordinates can be computed by the host computer, or displayed by the 4113 terminal.)

THE DIALOG AREA

The operator or the computer can create a *dialog area*: a part of the screen used for conversational dialogs between the operator and (a) the host computer or (b) the terminal. Text displayed in the dialog area does not interfere with pictures which the terminal may also be displaying.

TWO KINDS OF TEXT

The 4113 can display two kinds of text: *alphatext* and *graphtext*. Alphatext is used primarily for dialogs between the operator and the computer. Graphtext is used only in graphic displays.

Alphatext is the ordinary text displayed when the terminal is in alpha mode. If the optional APL keyboard is installed, then alphatext can be displayed in an alternate font used with the APL programming language. Figure 1-2 shows examples of alphatext.

Graphtext is text occurring within a <graphic-text> command. (The <graphic-text> command is described in Section 5, and in the 4110 Series Command Reference Manual.) Graphtext is used for fancy or high-quality lettering, especially within pictures or graphs. Graphic text may be displayed on the screen, or it may be included within the definition of a picture segment. However, it may not be displayed in the dialog area.

You can display graphtext in either of two degrees of "precision" (text quality): *stroke precision* or *string precision*. Stroke precision lets you rotate the text; it also lets you define your own character fonts. (Stroke precision is the default precision for graphtext.) Graphtext displayed with only "string precision" appears the same as alphatext. String-precision graphtext cannot be rotated, nor can you define your own string-precision text fonts.

Figure 1-3 shows examples of graphtext.

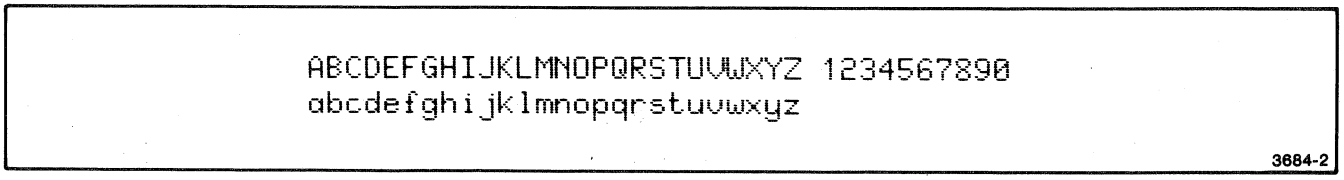


Figure 1-2. Samples of Alphatext.

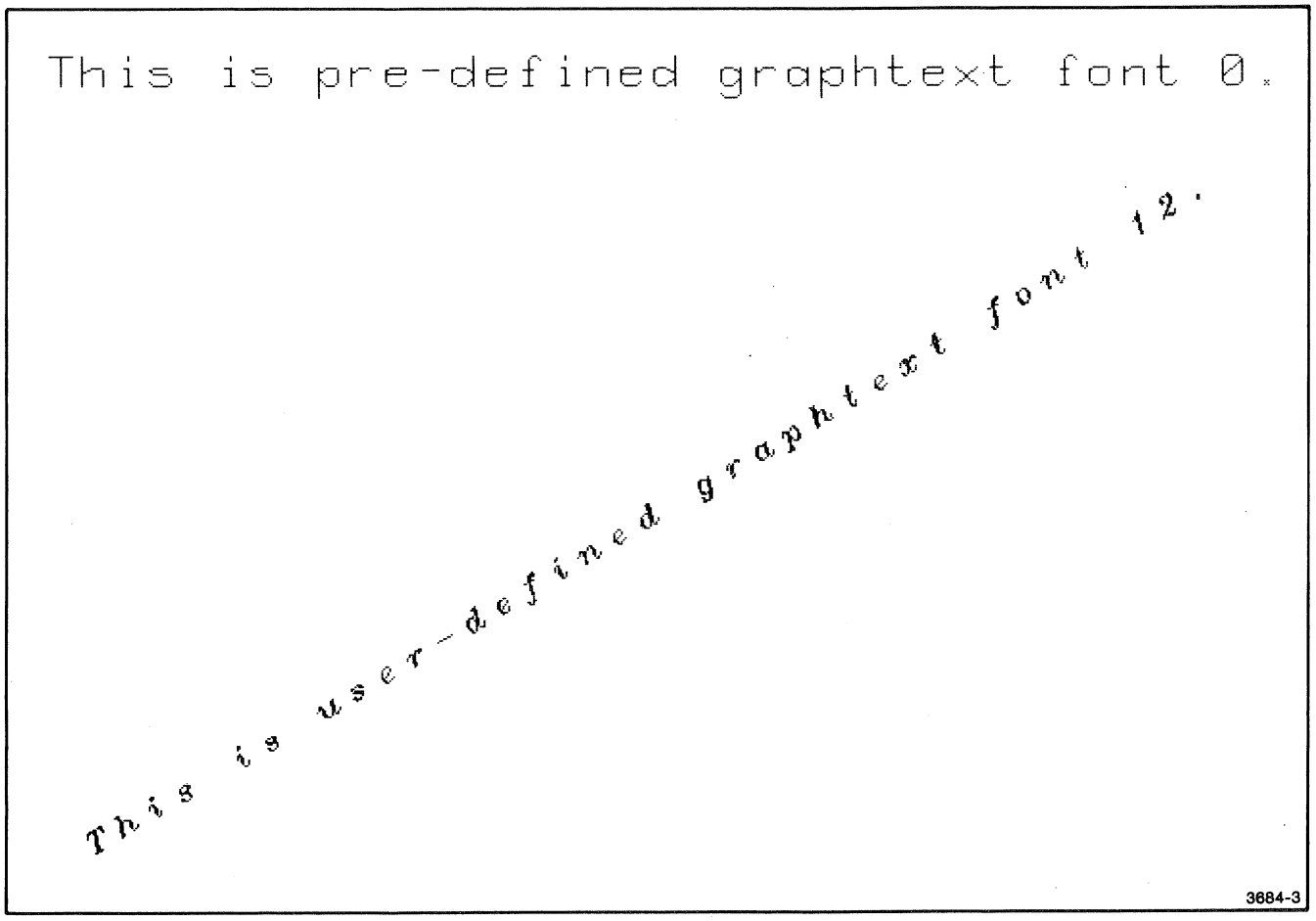


Figure 1-3. Samples of Graphtext.

Section 2

COMMAND SYNTAX

SYNTAX NOTATION

All commands to the 4113 terminal are ultimately sent as a sequence of individual ASCII characters. However, to describe each operation in terms of those individual characters would be both tedious and confusing. Therefore, a number of notational conventions are used throughout this manual and the 4110 Series Command Reference Manual.

SYNTAX RULES

In this manual, command syntax is represented according to the following rules.

Rule One. Individual ASCII characters sent to or from the terminal are enclosed in parentheses. (In the jargon of compilers and parsers, these are the "terminal symbols" of the 4113 command language.)

Examples:

(A)	The ASCII uppercase "A" character.
(a)	The lowercase "a" character.
(ESC)	"Escape" control character.
(DEL)	"Delete" character (also called "rub out").
(US)	"Unit separator" control character.
(SO)	"Shift out" control character.

The parentheses characters are an exception; to decrease confusion, they are represented as follows:

("(")	"Left parenthesis" character.
(")")	"Right parenthesis" character.

Rule Two. Expressions enclosed in "angle brackets" represent sequences of ASCII characters which might be sent to or from the 4113. (In the jargon of compilers and parsers, these are the "non-terminal symbols" of the "4113 command language.")

NOTE

You can find the exact syntax of any such "angle bracket" expression by referring to the 4110 Series Command Reference Manual.

For example, the following expressions, since they are enclosed in "angle brackets," represent syntactic constructs defined in the 4110 Series Command Reference Manual:

<set-baud-rates>	A command that sets the transmit and receive baud rates (data rates) for communicating with the host computer.
<int>	An integer number as encoded for transmission to the terminal.
<xy>	A pair of x- and y-coordinates as encoded for transmission to the terminal as a part of a command.

Rule Three. To represent specific examples of commands or other syntactic constructs, specific values are included between the angle brackets. For example:

<set-baud-rates: 1200, 1200>	A command to set receiving and transmitting data rates to 1200 baud (1200 bits/second).
<int: −35>	The number −35, as encoded for transmission to the terminal.
<xy: (0,100)>	The coordinate pair x= 0, y= 100, as encoded for transmission to the terminal.

The same notation may be used to express the meaning of parameters, especially in the more complicated syntax definitions. For instance, the meaning of the <set-baud-rate> command's two parameters can be expressed as follows:

<set-baud-rates> = (ESC)(N)(R)
<int+ : *transmit-rate*>
<int+ : *receive-rate*>

Here, the italicized expressions *transmit-rate* and *receive-rate* tell the meanings of the <int+> parameters.

Rule Four. In defining the syntax of a command, the equals sign (=) means "has the following syntax." For example:

<set-baud-rates> = (ESC)(N)(R)<int+><int+>

Rule Five. In syntax definitions, the word "or" means "or alternatively." For example:

<set-alphatext-font> = (ESC)(SI)
or (ESC)(SO)

Rule Six. In syntax definitions, square brackets delimit items which may be omitted. For example:

<int> = [<Hil>][<Hil>]<Lol>

This could be expressed in words as, "An <int> consists of zero, one, or two <Hil>s, followed by a <Lol>."

Rule Seven. Syntactic constructs which may be repeated any number of times are followed by three dots. For example:

<fill-pattern-definition> = <begin-fill-pattern>
[<pixel-def>...]
<end-fill-pattern>

This could be expressed in words as, "A <fill-pattern-definition> consists of a <begin-fill-pattern> command, followed by zero or more <pixel-def>s, followed by an <end-fill-pattern> command."

EXAMPLES

Since the <begin-segment> command is represented with "angle brackets," it has a syntax definition in the 4110 Series Command Reference Manual. In that manual, the <begin-segment> syntax is given as follows:

<begin-segment> = (ESC)(S)(O)<int>

This means that the <begin-segment> command is sent to the terminal as three ASCII characters — "escape," "uppercase S," and "uppercase O," followed by an <int> parameter.

Since the <int> parameter is represented with angle brackets, it too has a syntax definition in the Command Reference Manual.

A particular example of the <begin-segment> command is the <begin-segment: 1> command. That can be represented as follows:

<begin-segment: 1> = (ESC)(S)(O)<int: 1>

Referring to the Command Reference Manual's discussion of the <int> parameter type, notice that <int: 1> is the single ASCII character for the digit 1:

<int: 1> = (1)

Thus, you can expand the <begin-segment: 1> command as follows:

<begin-segment: 1> = (ESC)(S)(O)<int: 1>
= (ESC)(S)(O)(1)

Again, consider the <set-baud-rates: 600, 600> command. The 4110 Series Command Reference Manual gives the syntax for <set-baud-rates> as follows:

<set-baud-rates> = (ESC)(N)(R)<int+><int+>

Referring to the discussion of the Command Reference Manual on <int+>, you can expand the <set-baud-rates: 600, 600> command as follows:

<set-baud-rates: 600, 600>
= (ESC)(N)(R) <int+ : 600><int+ : 600>
= (ESC)(N)(R)(e)(8)(e)(8)

Thus, a <set-baud-rates: 600, 600> command is sent to the terminal as the following sequence of ASCII characters:

(ESC)(N)(R)(e)(8)(e)(8)

COMMAND SYNTAX

All commands to the 4113 are sent from the host computer as a sequence of ASCII characters. A few of these commands consist of a single character; some are comprised of two characters; however, most consist of three or more characters.

ONE-CHARACTER COMMANDS

The following commands consist of only one ASCII character. Like all commands, they are described fully in the Command Reference Manual.

<enter-alpha-mode> = (US)

<enter-vector-mode> = (GS)

<enter-marker-mode> = (FS)

TWO-CHARACTER COMMANDS

Most commands consist of *escape sequences* — sequences of ASCII characters beginning with the (ESC) character. A few of these commands consist of only two characters:

<enable-4010-GIN> = (ESC)(SUB)

<enter-bypass-mode> = (ESC)(CAN)

<page> = (ESC)(FF)

<report-4010-status> = (ESC)(ENQ)

<set-alphatext-font> = (ESC)< (SI) or (SO)>

<set-4014-line-style> = (ESC)<char>

<4010-hard-copy> = (ESC)(ETB)

COMMANDS OF THREE OR MORE CHARACTERS

Most of the 4113 commands are escape sequences of three or more characters. These commands take the following format:

1. The first character is (ESC). This serves as a "flag" to tell the 4113 that the following characters comprise a command for it.
2. The next two characters comprise an *op code* to identify the command.
3. After the op code there may be one or more *parameters* of the following types:

<int> or <int+>	A sequence of characters representing an integer number.
<int-array>	An array of <int> parameters, including a "count" for the array sent as an <int> parameter at the beginning. (See the discussion of <array> parameter types in the 4110 Series Command Reference Manual.)
<real>	A sequence of two <int> parameters which together represent a single real number: <real> = <int><int> (Exactly *how* these two <int>s represent a real number is described in the 4110 Series Command Reference Manual in the discussion of the <real> parameter type.)
<xy>	A sequence of characters representing x- and y-coordinates for some location on the 4113's screen.
<char>	A single ASCII character in the range from (SP) to (~): ASCII decimal equivalents from 32 to 126.
<char-array> or <string>	An array of <char> parameters, preceded by a count, expressed as an <int> parameter.

When the terminal is receiving the parameter for a command, it ignores any characters (a) which are not valid characters for the parameter being received, and (b) which are not the "command terminator" characters: (ESC), (US), (GS), and (FS). For instance, most ASCII control characters are ignored; thus (CR) characters or other interline characters can be inserted within the command's parameters with no ill effect. (This is useful if the parameter is a very long <string> or <int-array>.)

4. Finally, the command is "terminated" — it comes to an end. A command can be terminated in two ways:
   - The command ends when all its parameters have been sent to the terminal.
   - A command may be terminated early (before all its parameters have been sent) by sending any of the following characters:

     (ESC). A command ends with the (ESC) character that begins another command.

     (US), (GS), and (FS). A command ends whenever the terminal receives a (US), (GS), or (FS) character. These characters have their usual effects as <enter-alpha-mode>, <enter-vector-mode>, and <enter-marker-mode> commands.

Defaults for Missing Parameters

When a command is terminated early, the 4113 assigns default values to the missing parameters.

These defaults are usually:

- 0 for <int> parameters
- 0.0 for <real> parameters
- (0,0) for <xy> parameters
- (NUL) for <char> parameters
- an array of 0 elements for <array> parameters

Assigning (NUL) as the default for missing <char> parameters is an exception to the rule that <char> parameters must represent characters in the range from (SP) to (~).

NOTE

Some commands are exceptions. When those commands are terminated early, the 4113 assigns other defaults than those just listed. The only reliable way to determine the defaults used by a particular command is to consult the description of that command in the 4110 Series Command Reference Manual.

PARAMETER TYPES

Parameters for escape-sequence commands may be variables of several different data types. Each such data type has its own syntax and coding scheme. The parameter types are:

<int> and <int+>	Integer numbers are sent to the terminal as <int> or <int+> parameters. These two parameter types have the same syntax and the same coding scheme; they differ only in the range of valid values. <Int> parameters represent integers in the range from −32768 to + 32767. <Int+> parameters represent integers from 0 to + 65535.
<int-array> and <int+-array>	An array of integers consists of an <int> (or <int+>) telling how many items are in the array, followed by <int>s for each of the items in the array.
<real>	<Real> parameters represent numbers which can assume fractional values. Each <real> parameter consists of two <int> parameters. The first <int> represents a number, while the second <int> represents a power of two by which that number is to be multiplied. <Real> parameters can assume values from −32767.0 to + 32767.0.
<char>	<Char> parameters are individual ASCII characters in the range from (SP) to (~). (They have decimal equivalents in the range from 32 to 126.) They represent the displayable ASCII characters.
<string> or <char-array>	<String>s, or <char-array>s, consist of an <int> (or <int+>) telling how many characters are in the string, followed by <char>s for each of those characters.
<xy>	An (x,y) coordinate pair as encoded for transmission to the terminal. The <xy> parameter syntax is described in Section 5, and in the 4110 Series Command Reference Manual.

<INT> AND <INT+> PARAMETERS

The <int> and <int+> parameter types have the same packing scheme and the same syntax. These parameter types differ only in the range of valid values: −32768 to + 32767 for <int> parameters, and 0 to 65535 for <int+> parameters. The <int> and <int+> syntax is as follows:

<int> = [<Hil>] [<Hil>] <Lol>

<int+> = [<Hil>] [<Hil>] <Lol>

where

<Hil> = an ASCII character in the range from (@) to (DEL) — except that the character sequence (ESC)(?) may be used instead of (DEL).

<Lol> = an ASCII character in the range from (SP) to (?).

Example

Figure 2-1 shows the packing scheme, using the number + 31416 as an example.

1. The number to be sent is represented as a 16-bit signed binary numeral:

$$+31416_{10} = +0111101010111000_2$$

2. That binary numeral is arranged in groups of 6, 6, and 4 bits:

$$+ \; \underline{0\,1\,1\,1\,1\,0} \; \underline{1\,0\,1\,0\,1\,1} \; \underline{1\,0\,0\,0}$$

3. If the most-significant six bits are all zero, then the first <Hil> character may be omitted. In this case, they are not all zero, so they are used (together with a "tag" bit of 1) to form the first <Hil> character:

$$+ \; \boxed{0\,1\,1\,1\,1\,0} \; 1\,0\,1\,0\,1\,1 \quad 1\,0\,0\,0$$

$$\text{first } <\text{Hil}> = \boxed{1}\boxed{0\,1\,1\,1\,1\,0} = (\text{\^{}})$$

4. If the most-significant twelve bits are all zero, then BOTH <Hil> characters may be omitted. That is not the case in this example. The second <Hil> character is formed from the next least-significant six bits:

$$+ \quad 0\,1\,1\,1\,1\,0 \; \boxed{1\,0\,1\,0\,1\,1} \; 1\,0\,0\,0$$

$$\text{second } <\text{Hil}> = \boxed{1}\boxed{1\,0\,1\,0\,1\,1} = (\text{k})$$

5. The <Lol> character's least-significant bits are the four least-significant bits of the binary numeral. The fifth least-significant bit is 1 if the number is positive, and zero if it is negative. The two high-order bits ("tag bits") are "01" so as to make the <Lol> character fall in the range from (SP) to (?):

$$\boxed{+} \; 0\,1\,1\,1\,1\,0 \quad 1\,0\,1\,0\,1\,1 \; \boxed{1\,0\,0\,0}$$

$$<\text{Lol}> = \boxed{0\,1}\boxed{1}\boxed{1\,0\,0\,0} = (8)$$

6. The characters to be sent to the terminal, then, are (^)(k)(8):

$$<\text{int} : 31416> = (\text{\^{}})(\text{k})(8)$$

3675-2

Figure 2-1. Example of <Int> and <Int+> Packing Scheme.

Table 2-1 lists several examples of <int> parameters. (For a more complete list, see Appendix B.)

Table 2-1

EXAMPLES OF <INT> PARAMETERS

Number	<Int> Parameter
0	(0)
1	(1)
2	(2)
3	(3)
4	(4)
5	(5)
6	(6)
7	(7)
8	(8)
9	(9)
10	(:)
11	(;)
15	(?)
16	(A)(0)
17	(A)(1)
−1	(!)
−2	(")
−15	(?)
−16	(A)(SP)
−17	(A)(!)
1023	(DEL)(?) or (ESC)(?)(?)
1024	(A)(@)(0)
1025	(A)(@)(1)
−1024	(A)(@)(SP)
−1025	(A)(@)(!)

<INT-ARRAY> PARAMETERS

Some commands take <int-array> (or <int+ -array>) parameters. These consists of sequences of <int> (or <int+ >) parameters. The first <int> or <int+ > tells how many items are in the array. Subsequent <int> s represent the individual array items.

For instance the array of integers (1, 5, −1, 16) would be sent to the terminal as follows:

<int-array: (1,5,−1,16)

= <int: 4> {the count of 4}
<int: 1> <int: 5> <int: −1> <int: 16>

= (4) (1) (5) (!) (A)(0)

For more information on <int> and <int+ > parameters, see the description in the 4110 Series Command Reference Manual.

<REAL> PARAMETERS

A "real" number is a variable which may assume non-integer (that is, fractional) values. Real numbers between −32767.0 and + 32767.0 are sent to the terminal as <real> parameters. These consist of a pair of <int> s. The first <int> represents a number; the second <int> represents the power of two by which that number is to be multiplied.

For instance, the number 3.25 may be represented as 13 multipled by two raised to the power −2. Thus,

<real: 3.25> = <int: 13> <int: −2>
= (=) (")

For more information about <real> parameters, see the description in the 4110 Series Command Reference Manual. Included there is an example of a routine which sends <real> parameters to the terminal.

<CHAR> PARAMETERS

The <char> parameter type represents displayable ASCII characters. Each <char> parameter is a single ASCII character in the range from (SP) to (~). (The decimal equivalent of a <char> character is in the range from 32 to 126.)

<STRING> OR <CHAR-ARRAY> PARAMETERS

Strings, or arrays of displayable ASCII characters, are sent to the terminal as <char-array> parameters. Each such parameter consists of an <int> (or <int+>) telling how many items are in the array, followed by one <char> for each array item. For more information, see the description in the 4110 Series Command Reference Manual.

<XY> PARAMETERS

The <xy> parameter type represents spatial coordinates. <Xy> parameters are sent as a group of one to five ASCII characters. (The packing scheme used is the same as that used for earlier TEKTRONIX terminals.) The x- and y-coordinates in an <xy> parameter can range from 0 to 4095. For more details, see Section 5 and the 4110 Series Command Reference Manual.

REPORT PARAMETER TYPES

The parameter types described so far are for sending command parameters to the terminal. When the terminal sends messages back to the host computer, it packs the information in a different format. Thus, for each host-to-terminal parameter type there is a corresponding terminal-to-host parameter type. Table 2-2 lists the types.

Table 2-2

HOST-TO-TERMINAL AND TERMINAL-TO-HOST PARAMETER TYPES

Data to Be Sent	Host-To-Terminal Parameter Type	Terminal-To-Host Parameter Type
Integer (−32768 to +32767)	<int>	<int-report>
Integer (0 to 65535)	<int+>	<int-report>
Array of Integers	<int-array> or <int+ −array>	<int-array-report>
Real (−32767.0 to +32767.0)	<real>	<real-report>
Displayable Character	<char>	<char-report>
String of Characters	<string> or <char-array>	<string-report>
Spatial Coordinates	<xy>	<xy-report>

For more information on <int-report>s, <int-array-report>s, <real-report>s, <char-report>s, <string-report>s, and <xy-report>s, see the descriptions in the 4110 Series Command Reference Manual.

Section 3

THE PROGRAMMING TASK

This section provides an overview of the graphics programming task. Included are examples of how to use this manual to write device driver routines for the 4113.

Experienced programmers may find this section to be too elementary for them. These programmers should feel free, if they wish, to skip this section and proceed to Section 4.

GRAPHICS PROGRAM ARCHITECTURE

Figure 3-1 shows the architecture of a typical computer graphics program. The user's program consists of a main *applications program* and a *graphic subroutine package*. All graphic functions are performed through calls to subprograms in the graphics subroutine package. In turn, that subroutine package communicates (through host computer system software and the data communications line) with the 4113 terminal.

The graphic subroutine package includes *high-level graphics routines*, *device driver routines*, and *communications interface routines*.

It helps to define standard, uniform interfaces between these modules. For instance, a standard interface to the communications interface module lets you substitute different communications interface modules for use with different host operating systems. Likewise, you may wish to substitute different device drivers for use with different output devices (terminals, plotters, etc.).

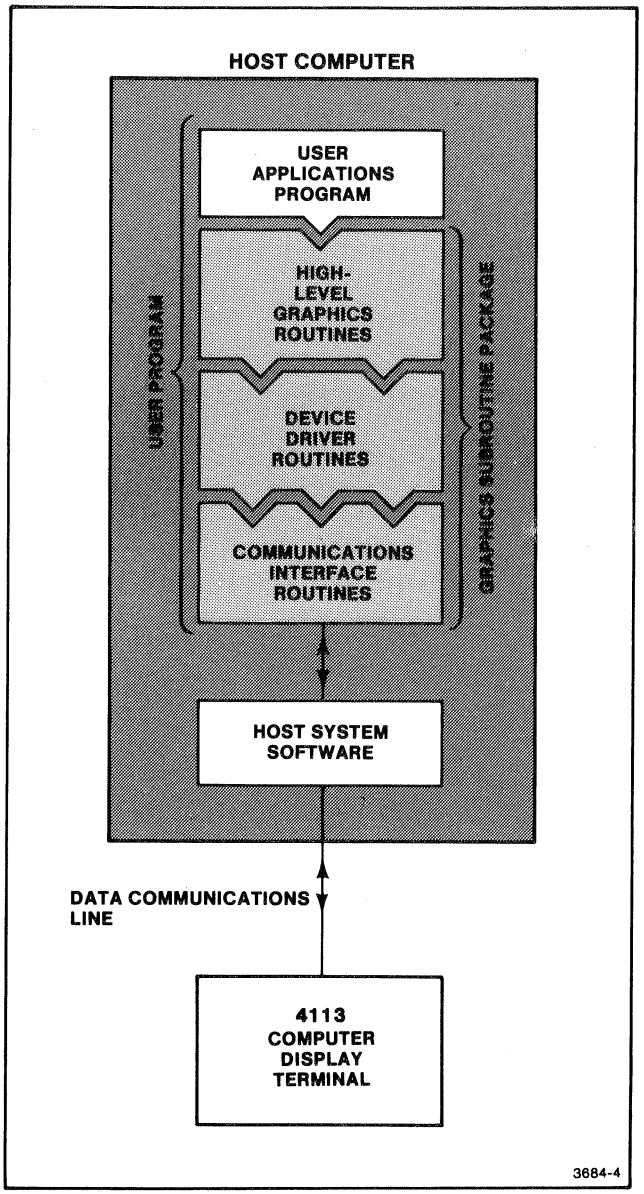


Figure 3-1. Graphics Program Architecture.

THE APPLICATIONS PROGRAM

The applications program is the main program; you write this program to perform the particular task you have in mind. The design of applications programs is beyond the scope of this manual.

GRAPHICS SUBROUTINE PACKAGE

There should be a standard, uniform interface between the applications program and the graphics subroutine package. (That way, you can use the same subroutine package with different applications programs.)

It is not good practice for the applications program to interface directly to the terminal, issuing commands to the terminal itself. Such a program is difficult to change for use with another graphics output device, such as a plotter or another terminal. Your applications program should interface to the terminal only by calls to standard routines in the graphics subroutine package.

For FORTRAN programmers, IGL (the TEKTRONIX 4010C01 PLOT 10 Interactive Graphics Library) is one such subroutine package.

Whatever graphics subroutine package you use (whether it be IGL or one which you write yourself), that package will typically be subdivided into *high-level graphics routines*, *device driver routines*, and *communications interface routines*.

References

For information on the TEKTRONIX 4010C01 PLOT 10 Interactive Graphics Library, see the following TEKTRONIX manuals:

- 4010C01 PLOT 10 Interactive Graphics Library User's Manual
- 4010C01 PLOT 10 IGL Option 4A Graphic Segments Support User's Manual

HIGH-LEVEL GRAPHICS ROUTINES

The high-level graphics routines provide the interface to the applications program. These routines perform functions such as selecting a graphic output device, moving graphic objects from place to place in the user's coordinate space, and manipulating graphic objects.

The graphics package may provide for performing these high-level functions on a variety of graphic output devices (different terminals, different plotters, etc.). In order to do this, the high-level graphic routines use calls to standard routines in device driver subroutine packages.

As far as it is possible, there should be identical interfaces between the high-level graphics routines and the device driver routines for different terminals or plotters.

References

Describing how to write high-level graphics routines and applications programs is beyond the scope of this manual. For more information on these subjects, you may wish to consult the following references:

- William M. Newman and Robert F. Sproull, *Principles of Interactive Computer Graphics*. Second Edition. McGraw-Hill Book Co., New York, 1979.
- Association for Computing Machinery, Special Interest Group on Graphics, Graphics Standards Planning Committee, "Status Report of the Graphics Standards Planning Committee." *Computer Graphics*, Volume 13, Number 3, August 1979. (Available from ACM, P.O. Box 12105, Church Street Station, New York, New York 10279.)

DEVICE DRIVER ROUTINES

The device driver subroutines are the ones which actually issue commands to the terminal. The more primitive of these routines each issues just one command. For instance, there may be a routine which causes the terminal to draw a line between two points on its screen; that routine probably calls more primitive routines to issue <enter-vector-mode> commands and <xy> parameters.

When writing device driver routines, you will definitely be referring to this manual and to the 4110 Series Command Reference Manual; writing device driver routines is discussed more fully later in this section, under "Issuing Commands to the Terminal" and "Parsing Messages From the Terminal."

COMMUNICATIONS INTERFACE ROUTINES

The device driver routines must have a way to send and receive individual ASCII characters. The techniques for doing this may be different on different host computers. To enhance portability from one host computer to another, therefore, it is wise to send and receive characters only through calls to standard communications interface routines.

This latter point is especially important if you will be using the 4113's block mode communications feature. While the terminal is in block mode (which requires Option 01), all characters sent to and from the terminal must pass through the communications interface routines which handle the details of the block mode protocol.

Another task of the communications interface routines is to ensure that no data is lost in the transmissions between the host computer and the 4113 terminal. In this regard, problems may occur (a) when sending commands to change the terminal's communications settings, and (b) when sending a large number of commands to the terminal.

Changing the Terminal's Settings

When sending commands to change the terminal's communications or display settings, you should pause after sending the commands before sending more data to the terminal. The pause lets the terminal finish executing the commands. If you do not pause, the first few characters of data may be processed incorrectly before the terminal's settings are changed.

How do you make the host program wait until the terminal is ready? Either by "handshaking," by putting your program in a "wait" state, or by sending a number of "no-op" characters. Handshaking is useful primarily for arming and disarming block mode, while waiting is preferable for entering prompt mode. Some computers allow programs to enter a "wait" state for a programmable amount of time. For other systems, transmitting no-op characters — such as (SYN) or (NUL) — for about half a second should suffice.

Handshaking

"Handshaking" means exchanging control signals between the host program and the terminal. Its purpose is to ensure that the terminal does not receive data while it is busy changing its communications settings, and that the terminal does not receive commands faster than it can execute those commands.

To handshake after sending commands to change the terminal's settings, send a <report-4010-status> command. (Other "report" commands can be used; but <report-4010-status> consists of only two characters: (ESC)(ENQ).)

When the terminal is finished changing its settings, it then executes the <report-4010-status> command and sends a <4010-status-report> to the host computer. The host receives the <4010-status-report>; only after reading this report message does it proceed to send more data to the terminal.

Figure 3-2 shows a FORTRAN routine which may be used to perform this "handshake" operation.

Preventing the Input Queue From Overflowing

If the average rate at which the host sends commands exceeds the average rate at which the terminal can execute those commands, then, sooner or later, the terminal's communications input queue will overflow.

The 4113 can display simple alphatext and graphics, continuously, at data rates up to 9600 bits per second. However, there are many commands (such as <include-copy-of-segment> or <load>) which take an indeterminate amount of time to execute. Thus, if commands come thick and fast, it is possible to overrun the terminal's communications input queue. Should this occur, data would be lost; the terminal would not execute all the commands it receives.

```
      SUBROUTINE HNDSHK

C Send <report-4010-status> command, (ESC)(ENQ).
C This routine uses the subroutine SENDCH. The SENDCH
C routine, given a number in the range from 0 to 127,
C sends the corresponding ASCII character to the terminal.
C Thus CALL SENDCH(27) sends the (ESC) character, and
C CALL SENDCH(5) sends the (ENQ) character.

      CALL SENDCH(27)
      CALL SENDCH(5)

C Read (and ignore) the <4010-status-report>. (This report
C consists of five characters, followed by a carriage return
C or other end-of-line indicator.)

      READ (ITTY,100) REPORT
100   FORMAT(A5)

C If the host computer provides an echo of characters which the
C the terminal sends, then the last character of the "echo"
C can serve as the terminal's <bypass-cancel-character>. In that
C case, the following statement can be omitted. This statment
C sends a (LF) as the <bypass-cancel-character>.

      CALL SENDCH(10)

C Once the report has been read, it is safe to proceed to other
C tasks, and perhaps send more characters to the terminal.

      RETURN
```

3684-5

Figure 3-2. A Handshaking Routine.

The terminal's prompt mode and flagging modes, and its optional block mode, provide a variety of handshaking techniques. Any of these techniques will prevent the terminal's input queue from overflowing. The user applications program need not concern itself with these handshaking techniques; they can be handled by routines in the device driver or communications interface modules. (See Section 10 for information on prompt mode and flagging, and Section 11 for information on block mode.)

Of course, the communications routines can also do handshaking, using (for instance) the <report-4010-status> command: (ESC)(ENQ). That is, the communications routines can set the terminal input queue size to N characters (<set-queue-size> command). Then, after sending N characters, those routines can send a <report-4010-status> command and input the <4010-status-report> that the terminal returns. This handshaking guarantees that the terminal's input queue is empty, ready to receive more characters.

ISSUING COMMANDS TO THE TERMINAL

ROUTINES WHICH ISSUE SEVERAL COMMANDS

The examples given in the later sections of this manual will give you clues on how to write higher-level device driver routines. For instance, Figure 3-3 shows one such example.

Thus, to initialize the terminal's dialog area in exactly the way shown in the Figure 3-3, you may write a device driver routine like that in Figure 3-4.

To prepare a dialog area, you might issue the following commands:

<set-dialog-area-lines: 40>	Up to 40 lines of the dialog area can be viewed at any one time.
<set-dialog-area-chars: 50>	Each line can hold up to 50 characters.
<set-dialog-area-position: (0,0)>	The "dialog viewport" is to be in the lower left corner of the screen.
<enable-dialog-area: 1>	Alphatext from the host will be be displayed in the dialog area rather than the graphic area.
<set-dialog-area-visibility: 1>	Make the dialog area visible.

3684-6

Figure 3-3. An Example With Several Commands.

In Figure 3-4, several procedures are invoked to change the terminal's dialog area settings. (Each of these procedures sends a single command to the terminal.) The Handshake procedure guarantees that the terminal's settings have been changed before the host sends more characters to the terminal. (This procedure is just a PASCAL version of the HNDSHK FORTRAN subroutine shown earlier in this section.)

ROUTINES TO ISSUE A SINGLE COMMAND

Of course, the procedure in Figure 3-4 only works if you also write device driver routines to issue each of the commands it invokes. To write these routines, you should consult the 4110 Series Command Reference Manual. For instance, that manual gives the <set-dialog-area-lines> syntax as follows:

<set-dialog-area-lines> = (ESC)(L)(L)<int>

From this syntax definition, you can write a routine like that in Figure 3-5.

```
PROCEDURE InitializeDialogArea;

   BEGIN
      SetDialogAreaLines(40);
      SetDialogAreaChars(50);
      SetDialogAreaPosition(0,0);
      EnableDialogArea(1);
      SetDialogAreaVisibility(1);
      Handshake;
   END; {** of InitializeDialogArea procedure **}
```

3684-7

Figure 3-4. A Device Driver Routine To Issue Several Commands.

```
PROCEDURE SetDialogAreaLines(NumberOfLines : INTEGER);

   BEGIN
      SendAscii(27); {** Send the (ESC) character **}
      SendAscii(76); {** letter L **}
      SendAscii(76); {** letter L **}
      SendInt(NumberOfLines);
   END; {** of SetDialogAreaLines procedure **}
```

3684-8

Figure 3-5. A Device Driver Routine Which Issues One Command.

ROUTINES TO ISSUE COMMAND PARAMETERS

The example in Figure 3-5 in turn calls two lower-level procedures. SendAscii issues a single character to the terminal (possibly using block mode, if the terminal is in block mode). SendInt sends a single <int> parameter to the terminal.

When writing routines to issue command parameters, you should again consult the 4110 Series Command Reference Manual. For instance, when writing a routine to send an <int> parameter to the terminal, you should consult that manual about the <int> parameter type.

SUMMARY

When writing higher-level device driver routines, which issue several commands to the terminal, see the examples given in other sections of this manual. When writing lower-level device driver routines, which issue single commands (or single parameters) to the terminal, see the 4110 Series Command Reference Manual.

PARSING REPORT MESSAGES FROM THE TERMINAL

Certain commands, such as <enable-GIN> or <report-terminal-settings>, cause the terminal to send a "report" message back to the host computer. When writing device driver routines to parse (read and accept data from) these reports, you must consult the 4110 Series Command Reference Manual. There you will find the exact syntax of each report message; use those syntax specifications when writing routines to parse the different report messages.

AN EXAMPLE

Suppose, for instance, that you are writing a routine to parse a <4010-status-report>. (This is the message which the terminal sends in response to the <report-4010-status> command.) According to the 4110 Series Command Reference Manual, this report has the following syntax:

```
<4010-status-report>
= <4010-GIN-status-report>
  or
  <4010-non-GIN-status-report>

<4010-GIN-status-report> = <4010-xy-report>
                           <EOM-indicator>

<4010-non-GIN-status-report>
= <4010-status-byte>
  <4010-xy-report>
  <EOM-indicator>
```

(The <4010-GIN-status-report> is used if the terminal has been enabled for GIN (graphic input) with an <enable-4010-GIN> command. Otherwise, the <4010-non-GIN-status-report> is used.)

A routine to parse <4010-status-report> may look like the one shown in Figure 3-6. In this example, a global flag variable (In4010GinMode) and other parsing routines are needed. Those routines would be written in a similar manner.

PARSING <EOM-INDICATOR>S

The syntax of many report messages includes <EOM-indicator>s. Sometimes these are "optional" — the terminal may or may not send them, depending on conditions listed in footnotes to the syntax of the particular report.

If the terminal is in block mode (which requires Option 01), then it sends each <EOM-indicator> by terminating the block and setting the end-of-message bit in that block's <block-control-bytes>. (See the description of block mode in Section 11 for details.) The host program's communications interface routines should handle the block mode protocol. The parsing routines in the host program's device driver module should never see the <EOM-indicator>s.

If the terminal is not in block mode, then when it sends an <EOM-indicator>, it sends the <EOM-indicator> as the current <EOL-string>. The <EOL-string> is a sequence of one or two characters, set by the most recent <set-EOL-string> command. The <EOL-string> is typically (CR) or (CR)(LF).

```
PROCEDURE ParseA4010StatusReport(VAR StatusByte : INTEGER;
                                 VAR Xy : XyType);
   BEGIN
   IF In4010GinMode {** a global flag ** }
      THEN {** parse a <4010-GIN-status-report> **}
         BEGIN
         {** Clear the StatusByte variable **}
            StatusByte := 0;
         ParseA4010XyReport(Xy);
         ParseAnEomIndicator;
         In4010GinMode := FALSE;
         END
      ELSE {** if NOT In4010GinMode **}
         BEGIN {** parse a <4010-non-GIN-status-report> **}
         ParseA4010StatusByte(StatusByte);
         ParseA4010XyReport(Xy);
         ParseAnEomIndicator
         END
   END; {** of ParseA4010StatusReport procedure **}
```

3684-9

Figure 3-6. Example of a Parsing Routine.

To make it easier to parse reports from the terminal, you should choose an <EOL-string> which consists only of ASCII control characters. These are ASCII characters — such as (CR) or (LF) — with numeric equivalents in the range from 0 to 31. The other parts of the report (the parts other than the <EOM-indicator>) typically do not include control characters. Thus, if the terminal is not in block mode, it can skip over any control characters it encounters.

SIGNATURE CHARACTERS

Most of the report messages which the terminal sends to the host begin with "signature characters." When several graphic input devices are enabled at the same time, the reports from different devices are marked with different signature characters. The signature characters indicate which type of report message is being sent and thus help you parse the report.

For more information on signature characters, see Sections 8 and 9.

Section 4

DISPLAY AND KEYBOARD SETTINGS

CONTROLLING THE DISPLAY

CONTROLLING EFFECTS OF (CR) AND (LF)

Normally, one must send the 4113 both (CR) and (LF) to cause its cursor to move to the start of a new line: (CR) to move the cursor left to the current margin, and (LF) to move it down one line. However, some hosts may send only (CR), or only (LF); to cope with these hosts, the 4113 has the <CRLF> and <LFCR> commands. Table 4-1 shows examples of these; for more information (such as the command syntax), see the descriptions of these commands in the 4110 Series Command Reference Manual.

Table 4-1

CONTROLLING THE EFFECTS OF (CR) AND (LF) CHARACTERS

Example	Description
<CRLF: 1> = (ESC)(K)(R)<int: 1> = (ESC)(K)(R)(1)	This command causes the 4113 to respond to a single (CR) character as if it were (CR)(LF).
<LFCR: 1> = (ESC)(K)(F)<int: 1> = (ESC)(K)(F)(1)	This command causes the 4113 to respond to a single (LF) as if it were (LF)(CR).
<CRLF: 0> = (ESC)(K)(R)<int: 0> = (ESC)(K)(R)(0) <LFCR: 0> = (ESC)(K)(F)<int: 0> = (ESC)(K)(F)(0)	Issuing both these commands disables the effect of any preceding <CRLF: 1> and <LFCR: 1> commands.

CONTROLLING THE KEYBOARD AND DEFINING MACROS

LOCKING THE KEYBOARD

The host computer can prevent the terminal's operator from typing inappropriate commands by locking the keyboard during critical operations. To do this, the host issues the <lock-keyboard> command:

<lock-keyboard> = (ESC)(K)(L)<int>

Here, the <int> is one to lock the keyboard and zero to unlock it. When the keyboard is locked, the KBD LOCK light turns on and typing on the keyboard only rings the bell. Table 4-2 shows how to issue the command.

The host computer can lock the 4113's four "viewing keys" independently of the rest of the keyboard. (These are the keys clustered together to the left of the thumbwheels.) To lock these keys, use the <lock-viewing-keys> command:

<lock-viewing-keys> = (ESC)(R)(J)<int>

The <int> parameter is one to lock the viewing keys, and zero to unlock them. (However, even after a <lock-viewing-keys: 0> command, the keyboard as a whole — including the viewing keys — may still be locked if a <lock-keyboard> command has been issued.)

Table 4-2

LOCKING AND UNLOCKING THE KEYBOARD

Example	Description
<lock-keyboard: 1> = (ESC)(K)(L)<int: 1> = (ESC)(K)(L)(1)	Locks the keyboard.
<lock-keyboard: 0> = (ESC)(K)(L)<int: 0> = (ESC)(K)(L)(0)	Unlocks the keyboard.

DEFINING MACROS (AND PROGRAMMING KEYS)

The <define-macro> command defines a "macro" - a sequence of characters that is referred to by a macro number. "Invoking" a macro means to use a macro in place of one of its representations. A macro is invoked in one or more of the following ways:

- The terminal receives an <expand-macro> command.
- The terminal receives a byte corresponding to the macro.
- The operator presses a key corresponding to the macro.

Macros are referenced by macro numbers in the following ranges:

-32768 through -32742	byte macros
-32740 through -32737	byte macros
-32608 through -32513	byte macros
-1	deletes all macros
0 through 143	key macros
144 through 32767	host macros

Key and host macros can be invoked with the <expand-macro> command. An <expand-macro> command specifies a particular macro number and invokes the corresponding macro. In effect, the terminal replaces the <expand-macro> command with the characters in the macro's definition and responds as if it received those characters in its input stream.

Key macros can also be invoked by pressing the key (or key combination) that produces the ASCII decimal equivalent equal to the macro number. For example, the ASCII decimal equivalent of the uppercase P is 80. If macro number 80 has been defined, pressing uppercase P invokes that macro. Function keys also correspond to macro numbers as described later.

Byte macros are invoked whenever the terminal receives a character that matches a defined macro number. A character is matched to a macro number by adding the character's decimal equivalent to -32768. For example, the EM character, whose decimal equivalent is 25, invokes macro -32743 (-32768 + 25). Note that several characters that might cause problems if defined as macros match invalid macro numbers. For example, ESC (ASCII decimal equivalent 27) matches -32741, an invalid macro number.

The <define-macro> command has this syntax:

<define-macro> = (ESC)(K)(D)<int><int-array>

Here, the <int> parameter names the macro being defined. The <int-array> is an array of numeric equivalents for the characters being programmed into the macro. For instance, you can program the "uppercase A" key to mean "Hi!" by issuing the following command:

<define-macro: 65, "Hi!">

= (ESC)(K)(D)
<int: 65>
<int-array: (72,105,33)>

= (ESC)(K)(D)
<int: 65>
<int: 3><int: 72><int: 105><int: 33>

= (ESC)(K)(D)
(D)(1)
(3) (D)(8) (F)(9) (B)(1)

= (ESC)(K)(D)(D)(1)(3)(D)(8)(F)(9)(B)(1)

In the example, the three ASCII characters (H)(i)(!) are represented in the <int-array> parameter by their decimal equivalents: 72, 105, 33. Since 65 is the ASCII decimal equivalent for the letter A, macro number 65 can be invoked by typing the ASCII character A on the keyboard.

This macro may also be invoked from the host by means of the <expand-macro> command:

<expand-macro: 65>
= (ESC)(K)(X)<int: 65>
= (ESC)(K)(X)(D)(1)

Macro numbers in the range from 128 to 143 correspond to the programmable function keys, as follows:

128 Function key F1
129 Function key F2
130 Function key F3
131 Function key F4
132 Function key F5
133 Function key F6
134 Function key F7
135 Function key F8
136 Function key S1 (SHIFT-F1)
137 Function key S2 (SHIFT-F2)
138 Function key S3 (SHIFT-F3)
139 Function key S4 (SHIFT-F4)
140 Function key S5 (SHIFT-F5)
141 Function key S6 (SHIFT-F6)
142 Function key S7 (SHIFT-F7)
143 Function key S8 (SHIFT-F8)

Host macros (numbered 144 through 32767) can only be invoked by the <expand-macro> command. Although it might seem easier to always use byte macros, since they are easier to invoke, you can have only 127 byte macros defined at any one time. Using host macros lets you use many more macros.

For more information, see the Command Reference Manual for descriptions of the <define-macro> and <expand-macro> commands.

<KEY-EXECUTE-CHARACTER>

Normally, when the operator presses a key which has been programmed (with the <define-macro> command), the characters programmed into the key are sent to the host computer, just as if the operator had typed those characters on the keyboard. This includes characters which comprise an escape-sequence command for the terminal; instead of executing such a command, the terminal sends the characters which comprise that command to the host computer.

The <key-execute-character> provides a way around this difficulty. You issue a <set-key-execute-character> command to designate one of the ASCII characters as the <key-execute-character>. Then, when programming a key (that is, when defining a macro numbered from 0 to 143), you can place <key-execute-character>s at the start and end of part of the text being programmed into the key. Later, when the operator presses that key, the characters between the <key-execute-character>s are not sent to the host computer. Instead, the terminal responds to those characters locally, as if it were receiving them from the host.

The <key-execute-character> within a macro has its special effect only if the macro is invoked by pressing a key. If the macro is invoked with the <expand-macro> command, then any <key-execute-character>s within it have no special effect.

EXAMPLES

Programming a Key to Type a Message to the Host

Suppose you want to program a function key to log you off the host computer, and that your host computer accepts the following characters as a command to log off:

(L)(O)(G)(O)(F)(F)(CR)

From an ASCII chart (Appendix A), you find that those characters have the following decimal equivalents:

(L)	76
(O)	79
(G)	71
(O)	79
(F)	70
(F)	70
(CR)	13

You can program function key F1 by sending the following command to the terminal:

<define-macro: 128, (76,79,71,79,70,70,13)>

= (ESC)(K)(D)
<int: 128>
<int-array: (76,79,71,79,70,70,13)>

= (ESC)(K)(D)
(H)(0)
(7) (D)(<) (D)(?) (D)(7) (D)(?) (D)(6) (D)(6) (=)

= (ESC)(K)(D)(H)(0)(7)(D)(<)(D)(?)
(D)(7)(D)(?)(D)(6)(D)(6)(=)

Programming a Key to Display a Message Locally

The following commands program function key F1 to display the message "HI THERE!" on the terminal's screen. (The message is *not* sent to the host computer; it is only displayed locally in the terminal.)

<set-key-execute-character: 126>
= (ESC)(K)(Y)<int: 126>
= (ESC)(K)(Y)(G)(>)

<define-macro: 128, "~HI THERE~">

= (ESC)(K)(D)
<int: 128>
<int-array: (126,72,73,32,84,72,69,82,69,126)>

= (ESC)(K)(D)
(H)(0)
<int: 10> <int: 126> <int: 72> ... <int: 126>

= (ESC)(K)(D) (H)(0) (:) (G)(>) (D)(8) (D)(9) (B)(0)
(E)(4) (D)(8) (D)(5) (E)(2) (D)(5) (G)(>)

The first command sets the <key-execute-character> to (~), which has an ASCII decimal equivalent of 126.

The second command defines macro number 128, which can be invoked by pressing function key F1. The numbers in this command's <int-array> are the ASCII decimal equivalents of the characters in the "~HI THERE~" message. Note that the message begins and ends with the (~) character, which is the current <key-execute-character>. When you press function key F1, the "HI THERE" message is executed (displayed) locally, as if it had come from the host computer.

Recall, however, that the <key-execute-character> has its special effect *only* when the macro is invoked by pressing the corresponding key (in this case, function key F1). If the macro is invoked by an <expand-macro: 128> command from the host computer, then the terminal behaves as if it had received the entire macro contents (including the (~) characters) from the host computer.

Programming a Key to Execute a Command Locally

You can program a key (define a macro) using the characters that comprise an escape-sequence command.

For example, one easy command is the <page> command, (ESC)(FF). You can program the <page> command into function key F2 as follows:

<define-macro: 129, (126,27,12,126)>

= (ESC)(K)(D)
<int: 129>
<int-array: (126,27,12,126)>

= (ESC)(K)(D)
<int: 129>
<int: 4><int: 126><int: 27><int: 12>
<int: 126>

= (ESC)(K)(D)
(H)(1)
(4) (G)(>) (A)(;) (<)
(G)(>)

Here, the <int-array> contains the numbers 126, 27, 12, 126; these are the ASCII decimal equivalents for the characters (~)(ESC)(FF)(~). The (~) characters are the <key-execute-character>s, while the characters between them comprise the <page> command.

DISPLAYING ALPHATEXT

ENTERING AND LEAVING ALPHA MODE

The 4113 Computer Display Terminal enters alpha mode whenever (a) it is turned on, or (b) it receives the <enter-alpha-mode> command: the character (US).

In addition, the <enable-dialog-area> command has an effect on other ways of entering alpha mode. (This command is described later in this section, in connection with the dialog area.) If the 4113 is emulating earlier 4010-Series TEKTRONIX terminals (dialog area not enabled), then it enters alpha mode whenever:

a. it receives a (CR) character,
b. it receives a <page> command, or
c. the operator presses the PAGE key.

The terminal leaves alpha mode on receiving an <enter-vector-mode> or <enter-marker-mode> command.

Tables 4-3 and 4-4 show how to put the 4114 in alpha mode and display a message on its screen. For each command, the table shows the individual ASCII characters sent to the terminal; also shown are high-level language (FORTRAN or PASCAL) statements to cause the host to send those characters to the terminal.

NOTE

The SENDCH subroutine (in Table 4-3) and the SendASCII procedure (in Table 4-4) accomplish the same purpose. Given the decimal equivalent of an ASCII character, these subprograms send that character to the terminal.

The "characters sent" in the two examples are those sent by the host computer (a DEC-system 10) used to test the examples.

In FORTRAN, the FORMAT statement has a "carriage control character" which generates a (LF) before the text; the FORMAT statement also puts a (CR) after the text. In PASCAL, on the other hand, the Writeln statement generates a (CR)(LF) sequence after the text. These differences are minor and should cause no concern.

Table 4-3

DISPLAYING ALPHATEXT USING FORTRAN

Command	Characters Sent	FORTRAN Statements
<enter-alpha-mode>	(US)	CALL SENDCH(31)
<alpha text: "Hi Mom!">	(LF)(H)(i)(SP)(M)(o) (m)(!)(CR)	WRITE(ITTY,100) 100 FORMAT(' Hi Mom!')

Table 4-4

DISPLAYING ALPHATEXT USING PASCAL

Command	Characters Sent	PASCAL Statements
<enter-alpha-mode>	(US)	SendAscii (31);
<alpha text : "Hi Mom!">	(H)(i)(SP)(M)(o)(m)(!)(CR)(LF)	Writeln(TTY,'Hi Mom!');

<SET-GRAPHIC-AREA-WRITING-MODE> COMMAND

The <set-graphic-area-writing-mode> command controls what happens when you backspace and type over text being displayed outside the dialog area. (The dialog area is described later in this section. There is a similar command, <set-dialog-area-writing-mode>, for text being displayed in the dialog area.)

There are two writing modes: "replace" mode (mode 0) and "overstrike" mode (mode 1). The terminal is shipped from the factory with the graphic area writing mode set to "overstrike."

In overstrike mode, alphatext characters are written over old characters without first erasing the old characters. That way, the operator or the host can underline characters by backspacing and typing over them with the "underscore" character, (_). This feature is useful with the APL character set (Option 4E), in which many "overstrike" character combinations are used.

In replace mode, each character erases any character which may formerly have been at that character position. This means that underlining is not possible. However, this writing mode is useful with the "line editing" features of some host operating systems.

The <set-graphics-area-writing-mode> command has this syntax:

<set-graphics-area-writing-mode>
= (ESC)(M)(G)<int>

The <int> is zero for replace mode, and one for overstrike mode. The operator can also typed this command in SETUP mode, using the the SETUP mode keyword, GAMODE. See the 4113 Operator's Manual for details.

THE DIALOG AREA

INTRODUCTION

The 4113 provides a *dialog area* for displaying alphatext. The dialog area is like a scroll of text, part of which is displayed in a part of the screen called the *dialog viewport*. It is possible, as described in Section 7, to display the dialog area in such a way that it does not interfere with graphics drawn on the screen.

The <enable-dialog-area> command controls whether or not alphatext is directed to the dialog area.

<Enable-dialog-area: 0> disables the dialog area; this makes the 4112 compatible with software written for earlier TEKTRONIX terminals which lack a dialog area. With the dialog area disabled, alphatext is directed to the screen (graphics area) and may be used, for instance, as labels on graphs.

<Enable-dialog-area: 1> enables the dialog area and directs alphatext to the dialog area. The alphatext is stored in an internal memory buffer (the *dialog scroll*). The operator can display the dialog scroll contents by pressing the DIALOG key, which makes a portion of the dialog scroll visible on the screen. Likewise, the host can issue a <set-dialog-visibility> command to make the dialog scroll visible.

When visible, the dialog area appears in a limited part of the screen called the *dialog viewport*.

<ENABLE-DIALOG-AREA> COMMAND

The <enable-dialog-area> command has the following syntax:

<enable-dialog-area> = (ESC)(K)(A)<int>

If the <int> is zero, the terminal emulates earlier TEKTRONIX terminals which lack a dialog area. Alphatext and graphics are both sent to the same destination within the terminal: the main graphics display area.

If the <int> is one, alphatext and graphics are sent to different destinations within the terminal. Graphics goes, as before, to the main graphics display area. Alphatext, however, is directed to the dialog scroll, *regardless of whether that scroll is currently visible.* To make the scroll visible, the DIALOG key or the <set-dialog-area-visibility> command must be used.

The <enable-dialog-area> command affects the operation of several other terminal features. With the dialog area disabled, these features emulate earlier TEKTRONIX terminals which lack a dialog area. With the dialog area enabled, these features operate in a way which is more appropriate for use with a dialog area.

Table 4-5 lists the differences between operation with and without the dialog area enabled. For more details, see the descriptions in the 4110 Series Command Reference Manual of the <enable-dialog-area>, <enable-4010-GIN>, <page>, and <renew-view> commands, and of the (CR) character.

Table 4-5

FEATURES AFFECTED BY THE <ENABLE-DIALOG-AREA> COMMAND

Feature	Effect With Dialog Area Disabled	Effect With Dialog Area Enabled
PAGE Key, <Page> Command	Erases the viewport for the current view. Redraws all visible segments. Exits the terminal from 4010 GIN mode. Resets line style to line style 1. "Homes" the graphic beam. Puts the terminal in alpha mode.	Erases the viewport for the current view. Redraws all visible segments.
<Renew-View> Command	Erases the viewport for the current view. Redraws all visible segments.	Erases the viewport for the current view. Redraws all visible segments.
(CR) Character	Puts terminal in alpha mode. Performs "carriage return" action. Resets line style to line style 1. Removes the terminal from 4010 GIN mode.	**If in alpha mode,** performs "carriage return" action for the alphatext cursor in the dialog area. **If in vector or marker mode,** does nothing.

<SET-DIALOG-AREA-VISIBILITY> COMMAND

The <set-dialog-area-visibility> command determines whether text stored in the dialog scroll is visible to the terminal's operator. This command has the following syntax:

<set-dialog-area-visibility> = (ESC)(L)(V)<int>

Here, the <int> is one to make the dialog area visible, or zero to make it invisible.

When the terminal is turned on, the dialog area is visible only if the dialog area is enabled. (The terminal "remembers" whether the dialog area was enabled when it was turned off.)

The operator can control dialog area visibility by the DIALOG key and the SETUP mode command DAVIS. For more details, see the 4113 Operator's Manual.

SETTING DIALOG AREA SIZE AND POSITION

When creating a dialog area, you set several parameters to control such things as its size and location. Each of the parameters has its own command.

The <set-dialog-area-chars> command sets the width of the dialog viewport (number of characters per line).

The <set-dialog-area-lines> command sets the height of the dialog viewport (number of lines of text which are visible at the same time).

The <set-dialog-area-buffer-size> command sets the size of the internal memory which holds the dialog area scroll.

The <set-dialog-area-position> command sets the position of the dialog viewport's lower left corner.

The <set-dialog-area-index> command sets the color-indices used to write text into the dialog area, to write the background for each character of that text, and to erase the dialog area. (See Section 7 for information on color-indices.)

The <set-dialog-area-surface> command determines on which writing surface the dialog area will be displayed. It is possible to place the dialog area on a different writing surface than the current viewport, so that text in the dialog area does not interfere with graphics being drawn in the current viewport. (This is described in Section 7, "The 4113 Display.")

The following commands show how to create a dialog area. For details on the syntax of these commands, see the 4110 Series Command Reference Manual.

<set-dialog-area-chars: 40>
<set-dialog-area-lines: 10>
<set-dialog-area-buffer-size: 30>
<set-dialog-area-position: (0,0)>
<enable-dialog-area: 1>
<set-dialog-area-visibility: 1>

Here, the dialog viewport will be 40 characters wide and 10 lines high. The dialog buffer size is made large enough to hold 1200 alphatext characters. (That is 30 full-size 40-character lines; if the lines are shorter, more lines can be stored in the dialog scroll.) The dialog viewport is positioned at the lower left corner of the screen. Finally, the <enable-dialog-area> command directs alphatext to the dialog area, and the <set-dialog-area-visibility> command makes the dialog area visible to the operator.

<SET-DIALOG-AREA-WRITING-MODE> COMMAND

The <set-dialog-area-writing-mode> command is similar to the <set-graphics-area-writing-mode> command, described earlier in this section. However, the <set-dialog-area-writing-mode> command applies only to text displayed in the dialog area. As with the other command, there are two writing modes, "overstrike" mode and "replace" mode. The terminal is shipped from the factory in overstrike mode.

The command syntax is as follows:

<set-dialog-area-writing-mode> = (ESC)(L)(M)<int>

The <int> parameter is zero for replace mode, and one for replace mode.

The operator can also type this command in SETUP mode, using the the SETUP mode keyword, DAMODE. See the 4113 Operator's Manual for details.

<SET-DIALOG-AREA-INDEX> COMMAND

The <set-dialog-area-index> command controls the color-indices used to display alphatext in the dialog area. For a description of this command, see Section 7, which describes color-indices.

USING THE OPTIONAL "APL" FONT

If your 4113 is equipped with Option 4E (the APL keyboard), then you may switch between two alphatext fonts by means of the <set-alphatext-font> command:

<set-alphatext-font: standard> = (ESC)(SI)

<set-alphatext-font: APL > = (ESC)(SO)

The character sequence (ESC)(SO) causes the terminal to switch to the APL character font for displaying alphatext. The character sequence (ESC)(SI) causes the terminal to switch back to the standard ASCII font.

Figure 4-1 shows the standard ASCII font and the optional APL font.

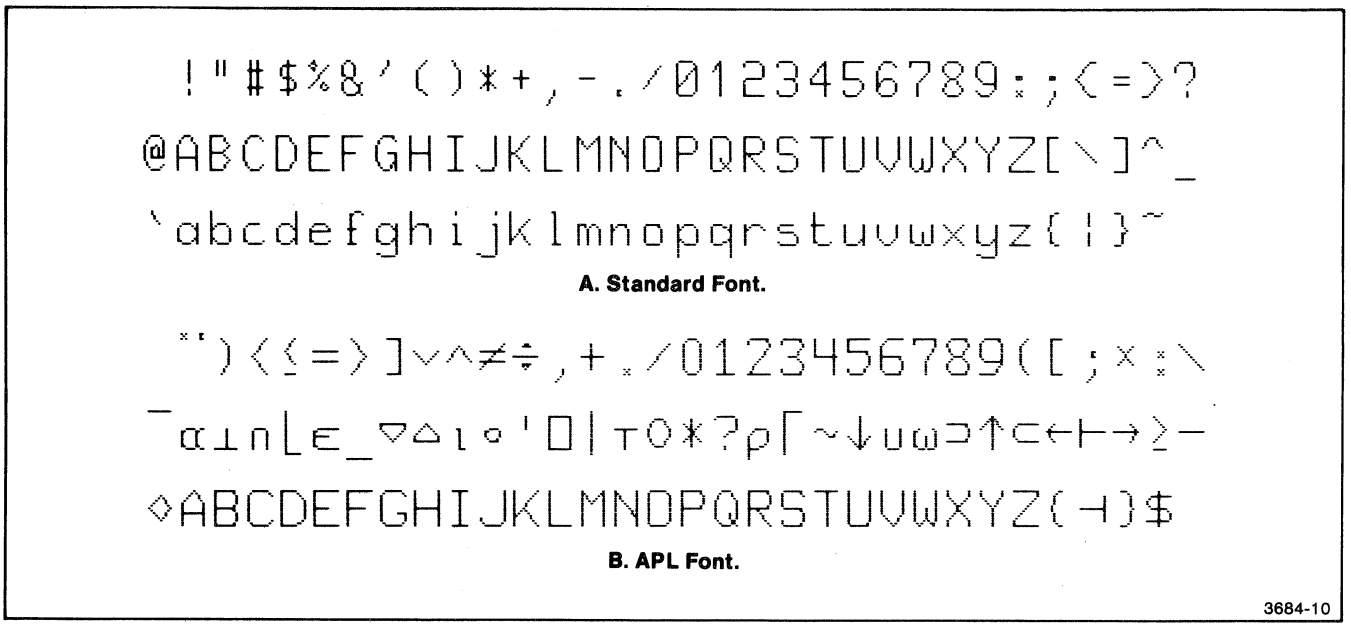


Figure 4-1. Standard and APL Character Fonts.

Section 5

DISPLAYING GRAPHIC INFORMATION

This section tells how to make the 4113 draw pictures on its screen. Topics here are:

- **Terminal Space Coordinates.** The coordinate system used to express the locations of graphic objects, and the <xy> coding scheme used to send terminal space coordinates to the 4113.
- **Drawing Lines.** How to draw line segments on the 4113's screen.
- **Attributes of Lines.** Varying how lines are displayed.
- **Markers.** Displaying markers (small, standard graphic objects).
- **Panels.** Closed regions whose interiors can be filled with various fill patterns.
- **Displaying Graphtext.** How to display text together with graphics on the 4113 display.
- **Defining Graphtext Characters.** How to define your own graphtext characters.

TERMINAL SPACE COORDINATES

Graphics in the 4113 are drawn in so-called "terminal space." This is a discrete two-dimensional space in which x- and y-coordinates are integers in the range from 0 to 4095. There are 16,777,216 addressable points in terminal space (4096 x 4096 = 16,777,216).

When sending graphic coordinates to the terminal, x- and y-coordinates can range from 0 to 4095. However, not all of these points in terminal space may be in view. When the terminal is first turned on, points with y-coordinates greater than 3127 cannot be seen. (See "Views," in Section 7, for more information on this topic.)

It is possible for the operator to "zoom" and "pan" around terminal space, changing the part of it which is in view on the screen. This is described in the 4113 Operator's Manual, and in Section 7 of this manual.

<XY> PARAMETERS

To send a pair of x- and y-coordinates to the terminal, you must encode them in a certain way. In this manual, the term <*xy*> refers to a pair of x- and y-coordinates as encoded for transmission to the terminal. The term <xy: (100,200)> refers to the coordinate pair (100, 200), as encoded for transmission to the terminal.

Format of <XY> Coordinate Bytes

Each <xy> parameter consists of one to five ASCII characters (seven-bit bytes). The bytes are sent in this order: <HiY> <Extra> <LoY> <HiX> <LoX>. Figure 5-1 shows the format of the five bytes.

Order and Meaning of the <XY> Bytes

1. The <**HiY**> (high-order y) byte comes first. This byte contains the most-significant five bits of the binary numeral representing the y-coordinate.

   0 1 y y y y y

   yyyyy : high-order five bits of the y-coordinate.

   You can omit the <HiY> byte if the high-order five bits of the y-coordinate have not changed since the last <xy> coordinate sent to the terminal.

2. Next comes the <**Extra**> byte. This byte contains the least-significant two bits of the x-coordinate, and the least-significant two bits of the y-coordinate.

   1 1 ? y y x x

   ? : "don't care" — may be either 0 or 1.
   yy : least-significant bits of y-coordinate.
   xx : least-significant bits of x-coordinate.

   You can omit the <Extra> byte if the least-significant bits of the x- and y-coordinates have not changed since the last <xy> coordinate sent to the terminal. If you do send the <Extra> byte, you must follow it with the <LoY> byte.

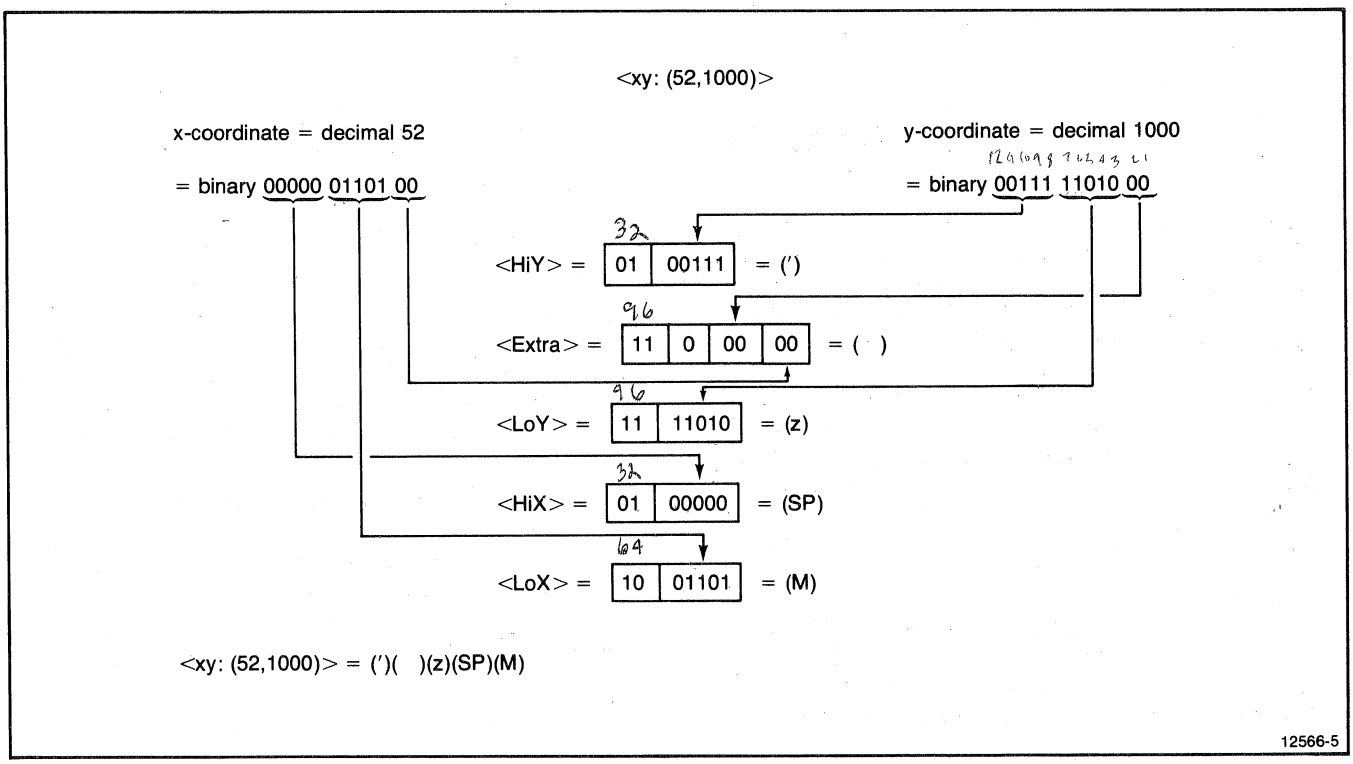


Figure 5-1. Format of <XY> Bytes.

3. Next comes the <**LoY**> (low-order y) byte. Despite its name, this byte contains the intermediate five bits of the 12-bit y-coordinate.

   1 1 y y y y y

   yyyyy : intermediate five bits of y-coordinate.

   You can omit the <LoY> byte provided: (a) you are sending neither the <Extra> byte nor the <HiX> byte in this <xy> coordinate, and (b) the intermediate five bits of the y-coordinate have not changed since the last <xy> coordinate sent to the terminal.

4. Next comes the <**HiX**> (high-order x) byte. This byte contains the high-order (most-significant) five bits of the x-coordinate.

   0 1 x x x x x

   xxxxx : most-significant five bits of x-coordinate.

   You can omit the <HiX> byte if the x-coordinate's most-significant bits have not changed since the last <xy> parameter sent to the terminal. If you do send the <HiX> byte, then you must precede it with the <LoY> byte.

5. Finally comes the <**LoX**> (low-order x) byte. Again, despite its name, this byte contains the intermediate five bits of the x-coordinate.

   1 0 x x x x x

   xxxxx : intermediate five bits of x-coordinate.

   This byte is always required, because it serves to terminate the <xy> byte sequence.

NOTE

The <LoY> and <Extra> bytes each have high-order bits of 11. Thus the (DEL) character (binary 1111111) is a possible <LoY> or <Extra> byte. Since some host computers use (DEL) as a filler character, this could be a problem in some installations.

The 4113 includes two features for overcoming this difficulty. First, it treats (ESC)(?) as a synonym for the (DEL) character. Secondly, it can be set to ignore (DEL) characters. (Use the <ignore-deletes> command, described in the 4110 Series Command Reference Manual.)

If your host computer uses (DEL) as a filler character, then you should use these features. Have the host send (ESC)(?) in place of (DEL) in <xy> parameters, and send an <ignore-deletes: 1> command to the terminal.

See the 4110 Series Command Reference Manual for more information on sending <xy> parameters to the terminal.

<XY> Syntax Summary

The following summarizes the formal syntax of <xy> coordinates:

<xy> = [<HiY>] [[<Extra>] <LoY> [<HiX>]] <LoX>

<HiY> = (SP) or (!) or (") or (#) or ($) or (%) or
(&) or (') or ("(") or (")") or (*) or (+) or
(,) or (-) or (.) or (/) or (0) or (1) or
(2) or (3) or (4) or (5) or (6) or (7) or
(8) or (9) or (:) or (;) or (<) or (>) or
(?).
{ ASCII characters with high-order bits "01".}

<Extra> = (') or (a) or (b) or (c) or (d) or (e) or
(f) or (g) or (h) or (i) or (j) or (k) or
(l) or (m) or (n) or (o) or (p) or (q) or
(r) or (s) or (t) or (u) or (v) or (w) or
(x) or (y) or (z) or ({) or (|) or (}) or
(~) or (DEL) or (ESC)(?).
{ ASCII characters with high-order bits "11".
and with (ESC)(?) as a synonym for (DEL).}

<LoY> = (') or (a) or (b) or (c) or (d) or (e) or
(f) or (g) or (h) or (i) or (j) or (k) or
(l) or (m) or (n) or (o) or (p) or (q) or
(r) or (s) or (t) or (u) or (v) or (w) or
(x) or (y) or (z) or ({) or (|) or (}) or
(~) or (DEL) or (ESC)(?).
{ ASCII characters with high-order bits "11",
and with (ESC)(?) as a synonym for (DEL).}

<HiX> = (SP) or (!) or (") or (#) or ($) or (%) or
(&) or (') or ("(") or (")") or (*) or (+) or
(,) or (-) or (.) or (/) or (0) or (1) or
(2) or (3) or (4) or (5) or (6) or (7) or
(8) or (9) or (:) or (;) or (<) or (>) or
(?).
{ ASCII characters with high-order bits "01".}

<LoX> = (@) or (A) or (B) or (C) or (D) or (D) or
(F) or (G) or (H) or (I) or (J) or (K) or
(L) or (M) or (N) or (O) or (P) or (Q) or
(R) or (S) or (T) or (U) or (V) or (W) or
(X) or (Y) or (Z) or ([) or (\) or (]) or
(^) or (_)
{ ASCII characters with high-order bits "10".}

CONSIDERATIONS WHEN SENDING <XY> COORDINATES TO THE TERMINAL

The terminal interprets <xy> coordinates correctly (rather than displaying them as alphatext) only if at least one of the following conditions is met:

- The <xy> coordinate is sent as a parameter for an "escape sequence" command.
- The terminal is in vector mode, having received an <enter-vector-mode> command (the (GS) character).
- The terminal is in marker mode, having received an <enter-marker-mode> command (the (FS) character).

The host program should, therefore, send <xy> coordinates to the terminal only when at least one of those conditions is met.

DRAWING LINES

There are two ways to draw lines on the screen. One way is by putting the terminal in vector mode and then sending <xy> coordinates to it. Another way is by means of <move> and <draw> commands.

VECTOR MODE

As mentioned in Section 1, the 4113 has three main modes of operation: alpha mode, vector mode, and marker mode. When the terminal is in alpha mode, it interprets alphanumeric characters coming from the host as letters and symbols to be displayed. In vector and marker modes, however, such characters are interpreted as <xy> coordinates representing points in terminal space. In vector mode, the terminal draws lines ("vectors") between these points.

Entering and Leaving Vector Mode

The terminal can enter vector mode from alpha mode, but not from marker mode. Therefore, if the terminal is in marker mode, you must first place it in alpha mode. You do this by sending a (US) character, which comprises the <enter-alpha-mode> command.

Once the terminal is in alpha mode, send the <enter-vector-mode> command: the single character, (GS).

To remove the terminal from vector mode, send it an <enter-alpha-mode> command (the (US) character) or an <enter-marker-mode> command (the (FS) character).

The Graphic Beam Position

The 4113 maintains a pointer in its memory to the current location in terminal space where it will display graphic information. This pointer is called the *graphic beam position*.

If the terminal is emulating earlier TEKTRONIX terminals which lack a dialog area (that is, if the dialog area is not enabled), then the graphic beam position also determines where alphatext will be displayed. (If the terminal enters alpha mode, the lower left corner of the next alphatext character will be at the graphic beam position.)

If the terminal is not emulating such earlier terminals (that is, if the dialog area is enabled), then alphatext is directed to the current cursor location in the dialog area scroll. In that case, the graphic beam position has no effect on alphatext.

Drawing Lines in Vector Mode

Once the terminal is in vector mode, it interprets any ASCII characters in the range from (SP) to (DEL) — decimal equivalents from 32 to 127 — not as characters to be displayed, but as <xy> parameters.

While in vector mode, the terminal ignores most of the ASCII control characters (characters with decimal equivalents less than 32). The exceptions are: (ESC), which signals the start of a new command; (BEL), which sounds the terminal's bell and makes the next vector a draw; the (US) and (FS) characters, which comprise <enter-alpha-mode> and <enter-marker-mode> commands; and — sometimes — the (CR) character. If the terminal is emulating certain earlier TEKTRONIX terminals (dialog area disabled), then the (CR) character puts the terminal back into alpha mode. If the terminal is not emulating such terminals (dialog area enabled), then it ignores any (CR) characters it receives while in vector mode.

Moving the Graphic Beam. While in vector mode, the terminal interprets an <xy> coming immediately after a (GS) character as a command to move the graphic beam to the point specified by the <xy> coordinates. The current graphic beam position is updated, but nothing is drawn on the screen.

Drawing Lines. Any subsequent <xy> parameter, *not* immediately following a (GS), causes the terminal to draw a line from the current graphic beam position to the point specified by the <xy> parameter. The beam position is updated so as to refer to the point at the end of the line just drawn.

The sequence (GS)(BEL)<xy> puts the terminal in vector mode, sounds the bell, and draws a line to the point specified by the <xy> coordinate. (This is described later in this section.)

To make the terminal "lift its pen" (move the beam without drawing anything), send another (GS). An <xy> coming immediately after a (GS) moves the graphic beam, while <xy>s that follow draw lines to the corresponding points.

An Example

Figure 5-2 shows commands which draw a square with diagonals. Table 5-1 expands these commands into the individual ASCII characters which are sent to the terminal.

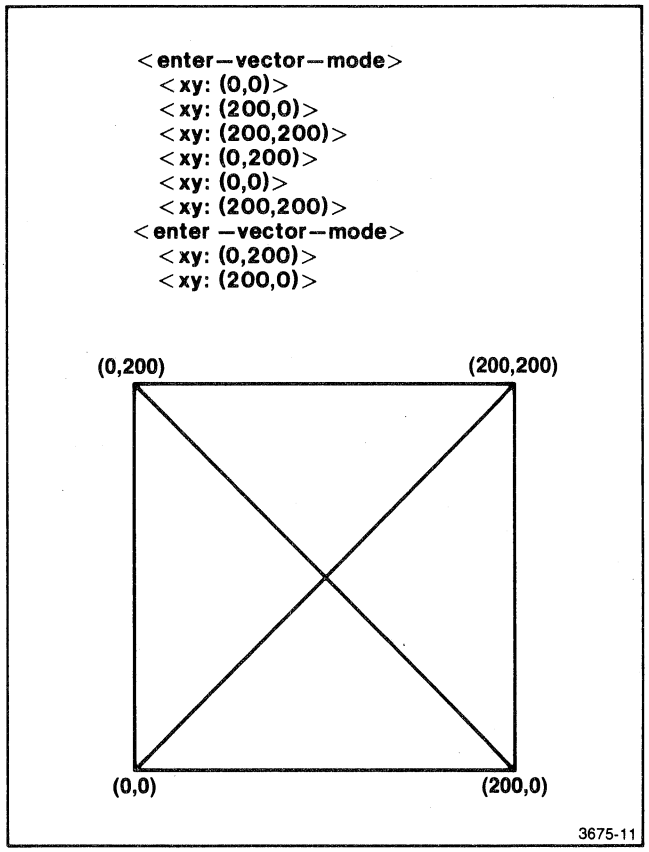


Figure 5-2. A Sample Figure Requiring "Moves" and "Draws."

Table 5-1

COMMANDS TO DRAW A SQUARE WITH DIAGONALS

Command	Characters Sent
<enter-vector-mode>	(GS)
<xy: (0,0)>	(SP)(`)(`)(SP)(@)
<xy: (200,0)>	(`)(!)(R)
<xy: (200,200)>	(!)(r)(R)
<xy: (0,200)>	(R)(SP)(@)
<xy: (0,0)>	(SP)(`)(@)
<xy: (200,200)>	(!)(r)(!)(R)
<enter-vector-mode>	(GS)
<xy: (0,200)>	(r)(SP)(@)
<xy: (200,0)>	(SP)(`)(!)(R)

Draws Without Moves: (GS)(BEL)

There are times when you may want to put the terminal in vector mode and then immediately draw a vector from the current beam position to some point — despite the fact that you do not know exactly where the graphic beam is positioned. To do this, precede the first <xy> parameter with a (BEL) character. The (BEL) prepares the terminal for a "draw" without changing the current beam position.

For example, in Figure 5-3 the graphic beam is moved to (0,1000), and an alphatext string is displayed there. Then a (GS)(BEL)<xy> sequence draws a line from where the next alphatext character would appear (if the terminal were to stay in alpha mode) to the point (2000,1000). (In this example, the <enable-dialog-area: 0> command disables the dialog area. This ensures that the alphatext is displayed together with the graphics, rather than being sent to the dialog area.)

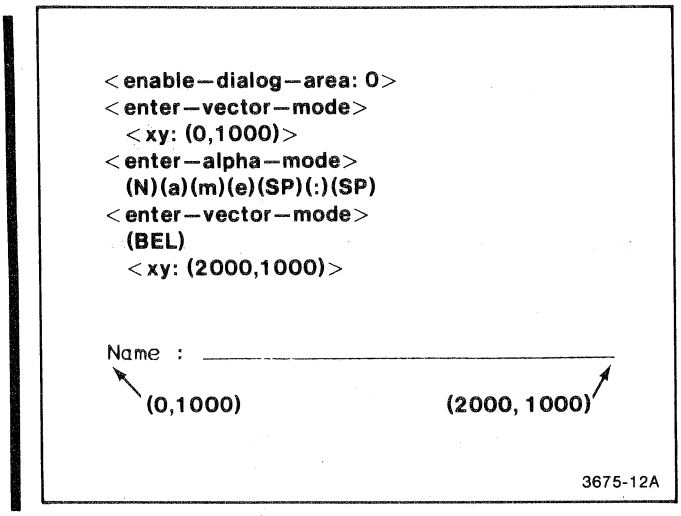


Figure 5-3. A "Draw" Without a Preceding "Move."

<MOVE> AND <DRAW> COMMANDS

Another way to draw lines is with the <move> and <draw> commands. These commands have the following syntax:

<move> = (ESC)(L)(F)<xy>

<draw> = (ESC)(L)(G)<xy>

The <move> command moves the graphic beam position to the point specified in its <xy> parameter. The <draw> command draws a line from the current graphic beam position to the point specified in its <xy> parameter.

Note that the <draw> command is like the (GS)(BEL)<xy> construct, in that it causes an immediate draw from any location.

These commands do not change the terminal's operating mode. If the terminal is in alpha mode when it receives a <move> or <draw> command, then it is still in alpha mode after the command has been executed. Likewise, if the terminal is in vector mode or marker mode on receiving a <move> or <draw>, then it is in that mode after executing the command.

ATTRIBUTES OF LINES

LINE STYLE

You can draw lines in different *line styles*: solid, dashed, etc. To do this, send the terminal a <set-line-style> command before drawing the lines. After that command, all subsequent lines are drawn in the line style specified by the command.

<Set-Line-Style> Command

The <set-line-style> command has the following syntax:

<set-line-style> = (ESC)(M)(V)<int>

Figure 5-4 shows examples of the different line styles.

<Set-4014-Line-Style> Command

The <set-4014-line-style> command is included for compatibility with software written for earlier TEKTRONIX terminals. This command is equivalent to the <set-line-style> command, but has a different syntax. See the 4110 Series Command Reference Manual for details.

LINE INDEX

You can control the line index (the color-index used for drawing subsequent lines) by sending a <set-line-index> command to the 4113. (color-indices are explained in Section 7.) The line index also affects how the line is drawn on an accessory plotter when a <plot> command is executed. (The <plot> command is described in Section 12, and in the 4110 Series Command Reference Manual.)

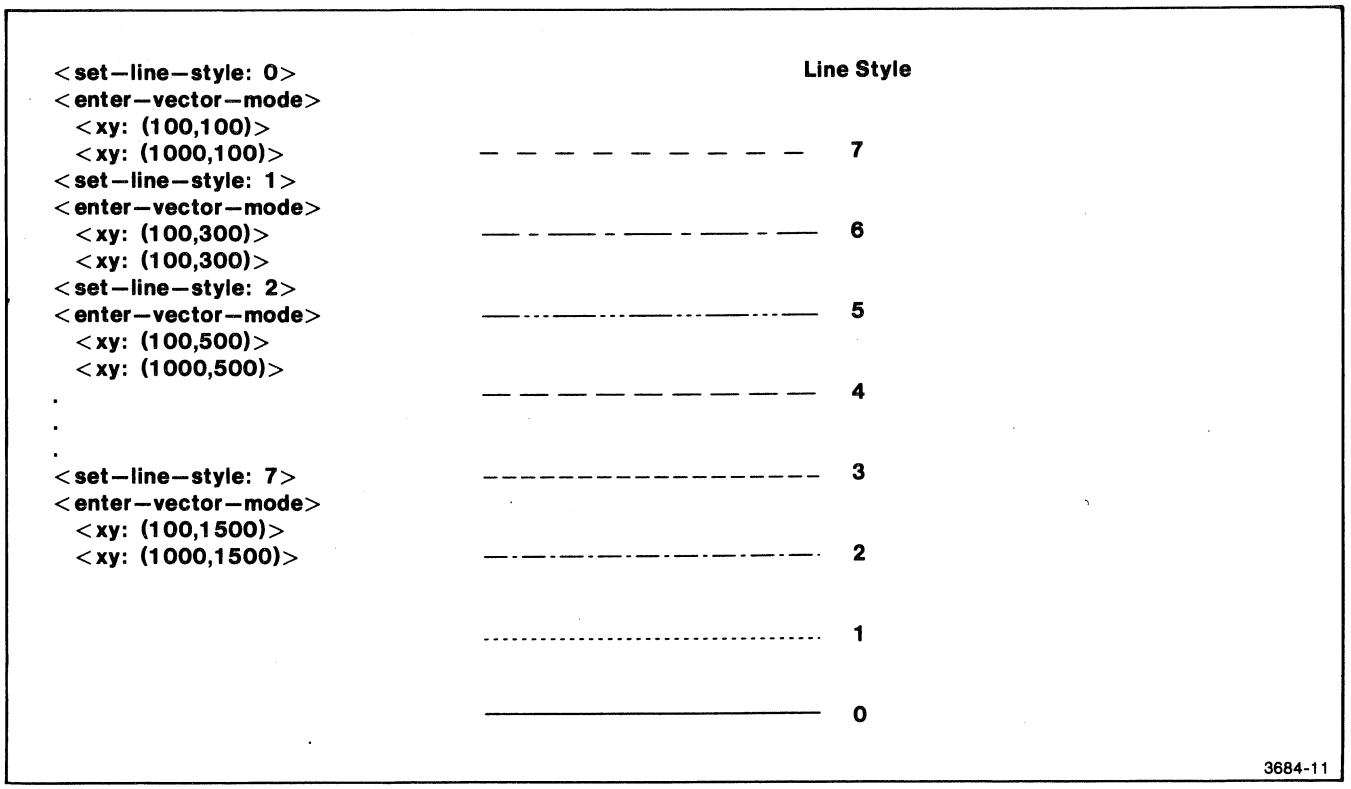


Figure 5-4. Line Styles Available in the 4113.

MARKERS

Markers are small, predefined graphic shapes which the 4113 can display anywhere in its 4096-by-4096 "terminal space." You can display markers in two ways: by putting the terminal in marker mode and sending <xy> coordinates, or by issuing individual <draw-marker> commands for each marker to be displayed.

MARKER MODE

One way to display markers is as follows:

1. Select the marker type to be displayed. To do this, send the terminal a <set-marker-type> command.
2. Send an <enter-marker-mode> command to the terminal. (This is just the single ASCII character, (FS).)
3. Send the terminal an <xy> coordinate for each marker to be displayed.
4. When done, issue an <enter-alpha-mode> command — the (US) character — to remove the terminal from marker mode.

<Set-Marker-Type> Command

The <set-marker-type> command has the following syntax:

<set-marker-type> = (ESC)(M)(M)<int>

Here, the <int> parameter may be in the range from zero to ten to select any of eleven predefined markers provided with the 4113.

Entering and Leaving Marker Mode

To put the terminal in marker mode, send it an (FS) character; this comprises the <enter-marker-mode> command.

While the terminal is in marker mode, (GS) characters (<enter-vector-mode> commands) have no effect. Therefore, the only way to remove the terminal from marker mode is to place it in alpha mode. This is usually done by sending a (US) character, which comprises the <enter-alpha-mode> command.

Example

Figure 5-5 shows commands which cause the terminal to display markers in each of the eleven predefined marker types.

THE <DRAW-MARKER> COMMAND

Another way to display a marker is to send the terminal a <draw-marker> command:

<draw-marker> = (ESC)(L)(H)<xy>

This causes the terminal to display one marker, of the current marker type, at the position specified by the command's <xy> parameter. The graphic beam position is updated to the position specified by the <xy> coordinate.

When the terminal has finished executing the <draw-marker> command, it returns to the mode it was in before executing that command. For instance, if the terminal was in alpha mode when it received the <draw-marker> command, it returns to alpha mode after displaying the marker — and likewise for vector mode and marker mode.

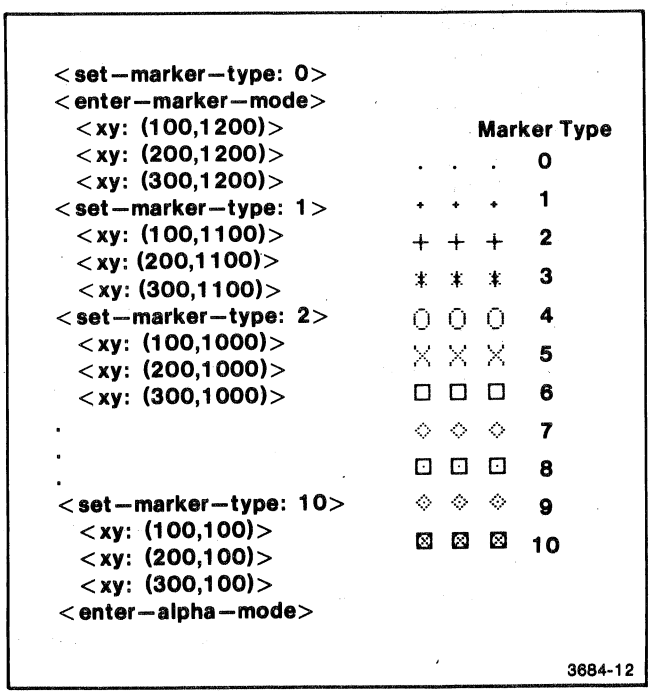


Figure 5-5. Displaying Markers.

DRAWING PANELS

The 4113 can draw *panels* — closed regions — and fill the interior of those panels with *fill patterns*.

PANELS WITH ONE BOUNDARY

Figure 5-6 shows commands to draw a simple panel and fill its interior with one of several predefined fill patterns provided with the 4113.

Here, the <select-fill-pattern> command chooses fill pattern number 15, one of 16 predefined fill patterns provided with the 4113. Other fill patterns can be used; but if a fill pattern number greater than 16 is chosen, then that pattern must previously have been defined. (Defining fill patterns is described in Section 7.)

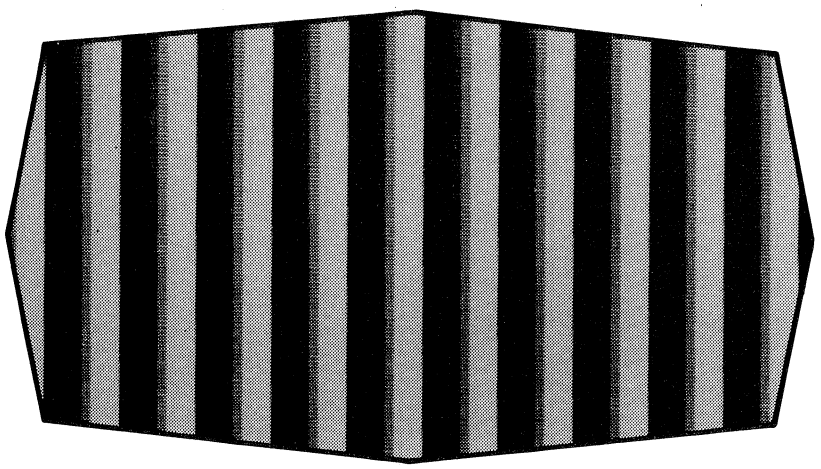
Fill patterns numbered from 0 to −15 cause the panel to be filled with pixels which are all the same color-index.

Selecting a fill pattern whose number is less than −15 causes the panel interior not to be filled.

The <set-panel-filling-mode: 0, 0, 1> command's three parameters specify how panels are to be filled. The first parameter selects "overstrike" rather than "replace" panel filling mode. (This is described in more detail later in this section.) The second parameter specifies that panels are to be filled up to, but not including, their boundaries. The third parameter specifies how the fill pattern in the panel interior is to be positioned. (Again, this is described later in this section.)

The <enter-vector-mode> command ensures that the terminal is in vector mode. (It is essential, during a panel definition, that the terminal be in vector mode or marker mode — *not* in alpha mode. This is necessary so that the terminal can correctly interpret the <xy> coordinates of the boundary points.)

```
<select-fill-pattern: 15>
<set-panel-filling-mode: 0, 0, 1>
<enter-vector-mode>
<begin-panel-boundary: (1000,1000), 1>
  <xy: (2000,900)>
  <xy: (3000,1000)>
  <xy: (3100,1500)>
  <xy: (3000,2000)>
  <xy: (2000,2100)>
  <xy: (1000,2000)>
  <xy: (900,1500)>
<end-panel>
```



3684-13

Figure 5-6. Drawing a Panel.

The <begin-panel-boundary> command starts the actual panel definition. This command has two parameters: an <xy> coordinate specifying the start (and end) of the panel boundary, and an <int> parameter specifying whether that boundary is to be drawn. This command has the following syntax:

<begin-panel-boundary> = (ESC)(L)(P)
<xy: *starting point*>
<int: 0 or 1>

The <int> parameter is one if the boundary is to be drawn, and zero if the boundary is not to be drawn.

If the boundary is drawn, it is drawn in the current line style and line index. All the boundary edges are drawn, regardless of any <enter-vector-mode> commands (that is, (GS) characters) that may be interspersed among the <xy> coordinates of its vertices.

After the <begin-panel-boundary> command, several <xy> coordinates specify points on the panel boundary.

NOTE

It is essential that the terminal should be in vector mode or marker mode while a panel is being defined. This is necessary in order that the terminal may properly interpret the <xy> parameters which specify the vertices of the panel boundary.

In this example, the <begin-panel-boundary> command specifies that the boundary is to be drawn. As the terminal receives the <xy> coordinates of the boundary points, it draws the boundary in the current line style and line index.

The panel definition ends with an <end-panel> command. When the terminal receives this command, it closes the panel boundary. That is, it draws a last line segment of the boundary, ending at the same point as was specified in the <begin-panel-boundary> command. The graphic beam position is updated to that point. Then the 4113 fills the panel interior, using the current fill pattern.

SELF-INTERSECTING PANEL BOUNDARIES

If a panel boundary crosses over itself, it is not obvious just which areas are "inside" the panel (and are therefore to be filled with the current fill pattern). The rule for determining which points are "inside" the panel is as follows:

- If, to get from "well outside the panel" (that is, from a point infinitely far away) to the point in question, one must cross the panel boundary an *odd* number of times, then that point is "inside" the panel.
- If, to get from well outside the panel to the point in question, one crosses the boundary an *even* number of times, then that point is "outside" the panel.

Figure 5-7 shows commands to draw a panel with a boundary which crosses over itself. For variety, a different fill pattern has been selected.

MULTIPLE PANEL BOUNDARIES

A panel can have more than one boundary. Figure 5-8 shows two such panels. Each panel boundary ends with the <begin-panel-boundary> command that starts the next boundary. The last boundary ends with the <end-panel> command that terminates the panel definition.

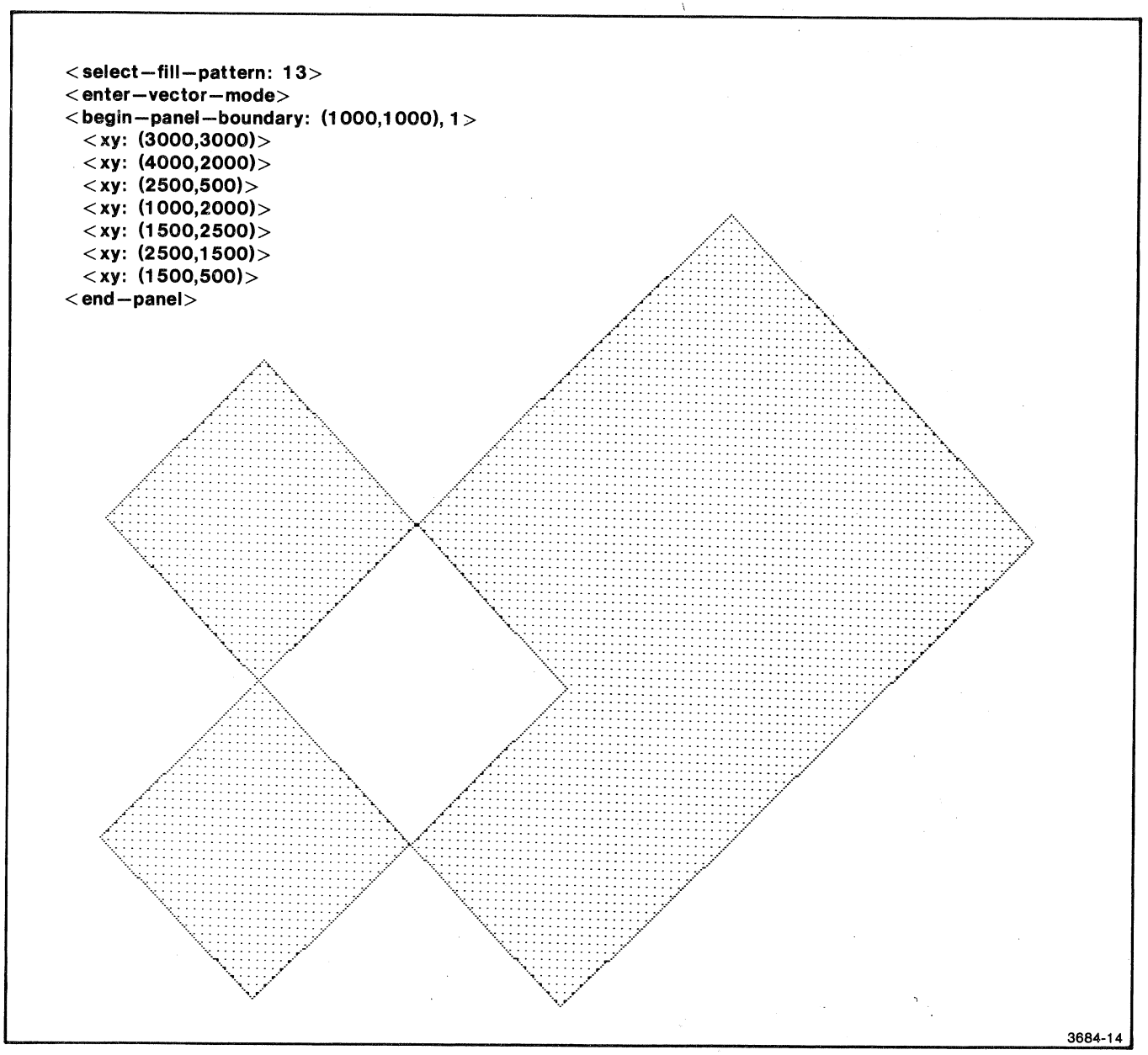
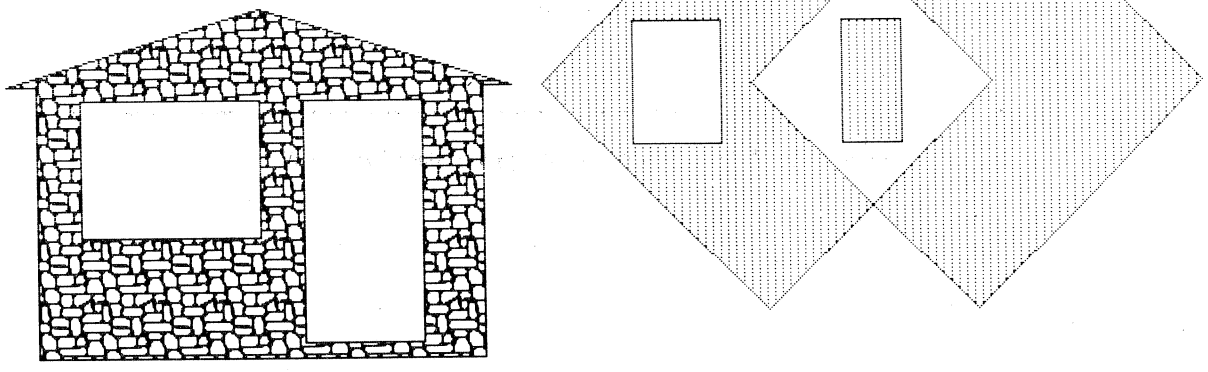


Figure 5-7. A Panel Whose Boundary Crosses Over Itself.

```
<select-fill-pattern: 16>
<enter-vector-mode>
<begin-panel-boundary: (100,100), 1>
  <xy: (1600,100)>
  <xy: (1600,1000)>
  <xy: (1700,1000)>
  <xy: (850,1250)>
  <xy: (0,1000)>
  <xy: (100,1000)>
<begin-panel-boundary: (250,500), 1>
  <xy: (850,500)>
  <xy: (850,950)>
  <xy: (250,950)>
<begin-panel-boundary: (1000,150), 1>
  <xy: (1400,150)>
  <xy: (1400,950)>
  <xy: (1000,950)>
<end-panel>

<select-fill-pattern: 14>
<begin-panel-boundary: (1800,1000), 1>
  <xy: (2550,250)>
  <xy: (3300,1000)>
  <xy: (2550,1750)>
<begin-panel-boundary: (2500,1000), 1>
  <xy: (3250,250)>
  <xy: (4000,1000)>
  <xy: (3250,1750)>
<begin-panel-boundary: (2100,800), 1>
  <xy: (2400,800)>
  <xy: (2400,1200)>
  <xy: (2100,1200)>
<begin-panel-boundary: (2800,800), 1>
  <xy: (3000,800)>
  <xy: (3000,1200)>
  <xy: (2800,1200)>
<end-panel>
```



3684-15

Figure 5-8. Two Panels, Each With Multiple Boundaries.

CONSIDERATIONS IN PANEL DEFINITIONS

Figure 5-9 summarizes the syntax for panel definitions.

Within a <panel-boundary-definition>, the <boundary-points> may be specified by <move> or <draw> commands, which have <xy> parameters. Or, a boundary point can be specified by an <xy> parameter which is not part of another command — but only if the terminal is in vector mode or marker mode. (If the terminal is in alpha mode, it does not interpret <xy> parameters correctly, but displays them as alphatext.)

If you wish, you can intersperse (GS) and (FS) characters (<enter-vector-mode> and <enter-marker-mode> commands) among the <xy> coordinates within a <panel-boundary-definition>. These serve to guarantee that the terminal is not in alpha mode.

Except for that guarantee, however, the panel boundary is unaffected by (GS) and (FS) characters interspersed among the <xy> coordinates. If the panel boundary is drawn at all, all of its edges are drawn, regardless of any interspersed (GS) character. No markers are drawn at the vertices of the boundary, regardless of any interspersed (FS) characters.

After the <end-panel> command, the terminal is in vector mode, marker mode, or alpha mode, according to the mode in which it was last placed. (It is, of course, bad practice to put the terminal in alpha mode within a <panel-definition>.) The graphic beam position is updated to the point specified by the <xy> parameter in the last <begin-panel-boundary> command.

<panel—definition> = <panel—boundary—definition> ...
<end—panel>

<panel—boundary—definition> = <begin—panel—boundary>
[<boundary—point> ...]

<boundary—point> = <xy> **(See Note.)**
or <enter—vector—mode><xy>
or <enter—marker—mode><xy>
or <move>
or <draw>
or <draw—marker>

<begin—panel—boundary> = (ESC)(L)(P)<xy><int>

<end—panel> = (ESC)(L)(E)

<enter—vector—mode> = (GS)

<enter—marker—mode> = (FS)

<move> = (ESC)(L)(F)<xy>

<draw> = (ESC)(L)(G)<xy>

<draw—marker> = (ESC)(L)(H)<xy>

Note. An <xy> is valid here only if the terminal has already been placed in vector mode or marker mode.

3684-16

Figure 5-9. A Valid <Panel-Definition> Syntax.

<SELECT-FILL-PATTERN> COMMAND

The <select-fill-pattern> command has the following syntax:

<select-fill-pattern> = (ESC)(M)(P)<int>

Here, the <int> specifies the pattern number to be used for filling subsequent panels. If this number is zero, panels are not filled. Numbers from 1 to 16 select predefined fill patterns, shown in Figure 5-10. Numbers from 16 to 32767 select fill patterns which the user has defined. (Defining fill patterns is described in Section 7.) Numbers 0 through −15 select fill patterns which are all one color-index, while numbers −16 and below cause the panel not to be filled.

If Option 21 (four bit planes of display memory) is installed, then sixteen different color-indices are available. For instance, fill pattern −2 consists of pixels which are all set to color-index 2. (On power-up, lines drawn in index 2 are red in color.)

(Bit planes and surfaces are described in Section 7.)

Defining Your Own Fill Patterns

You can define your own fill patterns, using the <raster-write> and <runlength-write> to specify the color-indices of the individual pixels in the fill pattern.

To define fill patterns, you must understand the concepts of color-indices in the 4113's raster memory space. Therefore, a description of how to define fill patterns is deferred to Section 7, where color-indices and color mixtures are explained.

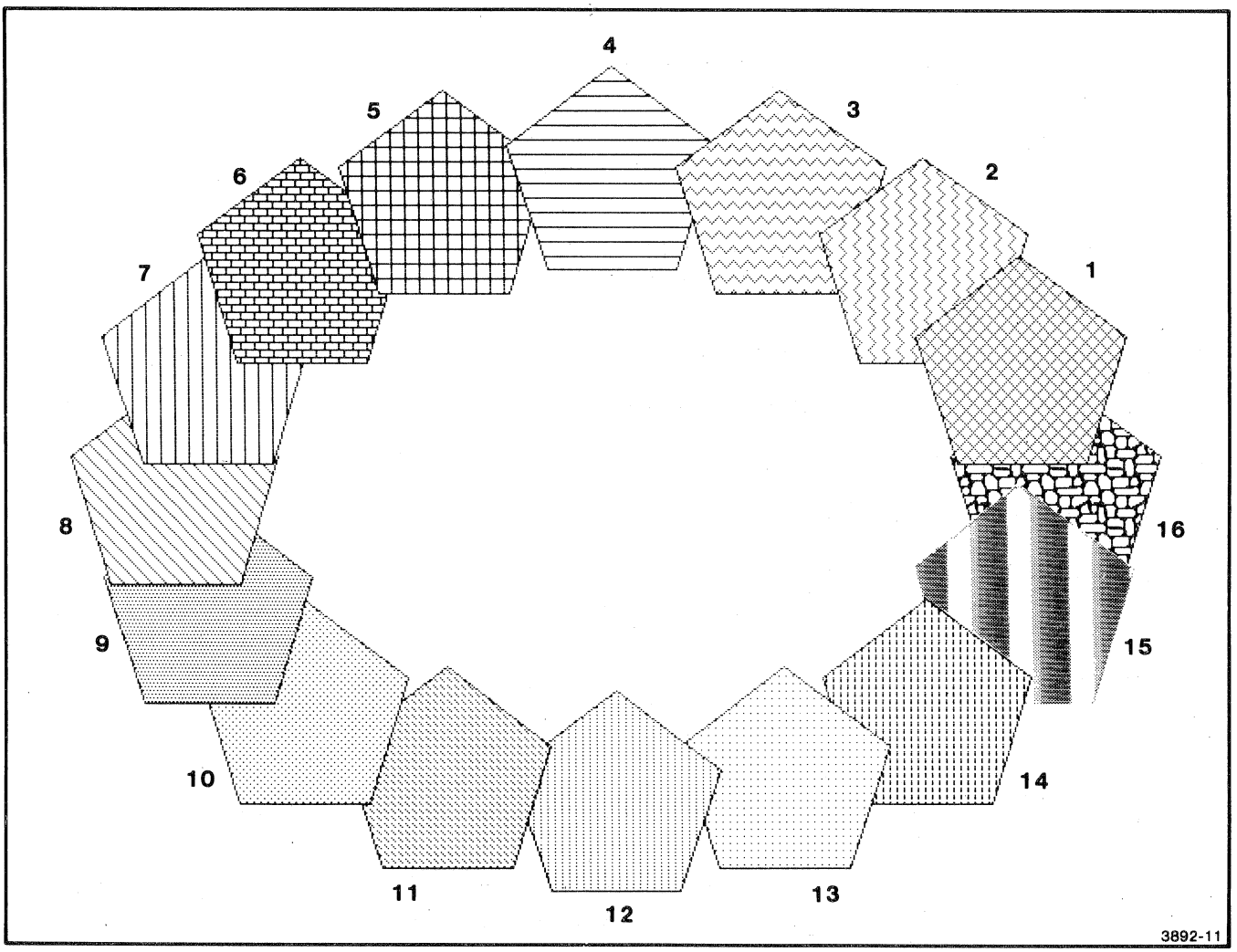


Figure 5-10. Standard Predefined Fill Patterns.

<SET-PANEL-FILLING-MODE> COMMAND

The <set-panel-filling= mode> command has the following syntax:

<set-panel-filling-mode>
= (ESC)(M)(S)
  <int: *overstrike-or-replace*>
  <int: *overlap-boundary*>
  <int: *keying-mode*>

where:

<int: *overstrike-or-replace*> = (0) or (1)

<int: *overlap-boundary*> = (0) or (1)

<int:*keying mode*> = (0) or (1) or (2) or (3)

The "overstrike or replace" parameter specifies the effect of fill pattern pixels which are in color-index zero. If this parameter is zero, panels are filled in replace mode. In that mode, zero pixels in the fill pattern cause the corresponding pixels in the panel interior to be set to color-index zero. If the overstrike/replace parameter is one, panels are filled in overstrike mode. In that mode, zero pixels in the fill pattern cause the corresponding pixels in the panel interior to be left unaltered. Figure 5-11 shows the effect. (See Section 7 for more information on color-indices.)

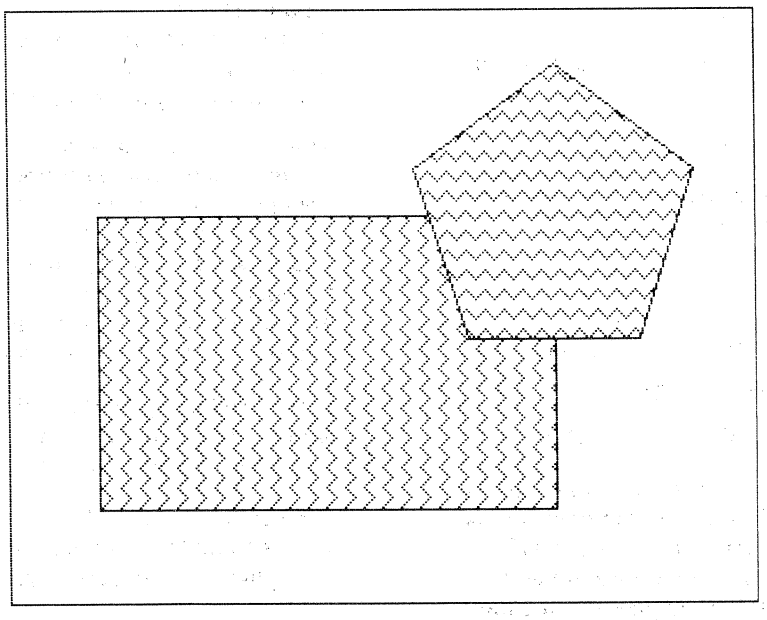
The second <int> parameter specifies whether, when a panel is filled, the fill pattern extends out to include the panel boundary. If this parameter is zero, the boundary is not filled. If the parameter is one, the boundary is filled. Figure 5-12 shows the effect.

The third <int> parameter specifies the *pattern keying mode.* In pattern keying mode one, fill patterns are positioned with respect to the viewport on the terminal's screen. (See Section 7 for an explanation of views and viewports.) Thus, if adjacent panels in the same viewport have the same fill pattern, that fill pattern is so aligned as to make the panels blend smoothly together. Figure 5-13A shows the effect.

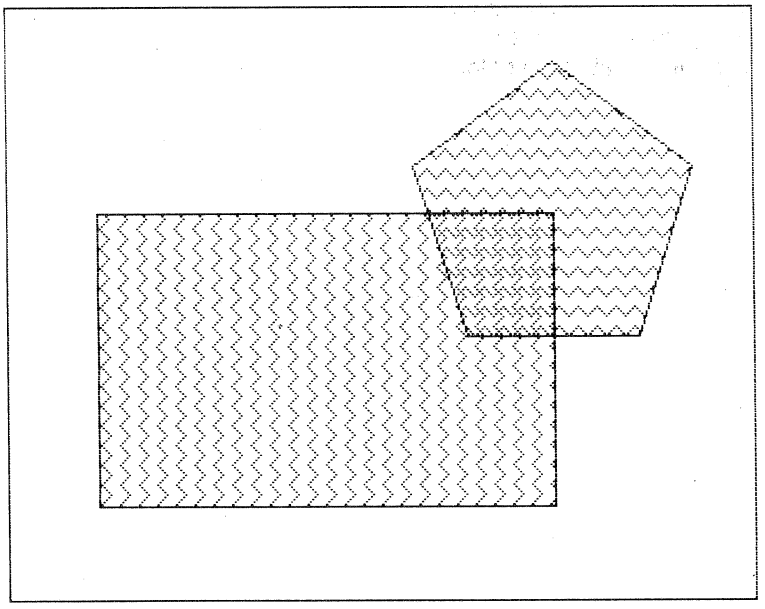
In pattern keying mode 2, fill patterns are positioned with respect to the panels themselves, rather than the viewport. Figure 5-13B shows the effect.

In pattern keying mode 3, fill patterns are positioned with respect to the entire display screen, rather than to the individual viewport. Figure 5-13C shows the effect.

If this parameter in the <set-panel-keying-mode> command is zero, then the pattern keying mode is left unchanged from its previous value.



A. Replace Mode.

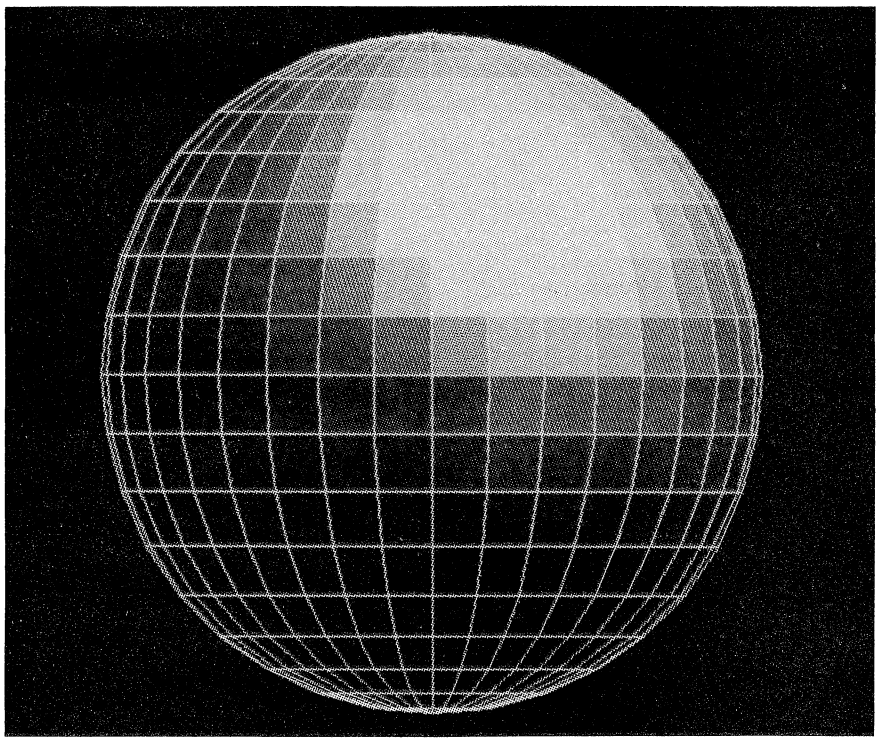


B. Overstrike Mode.

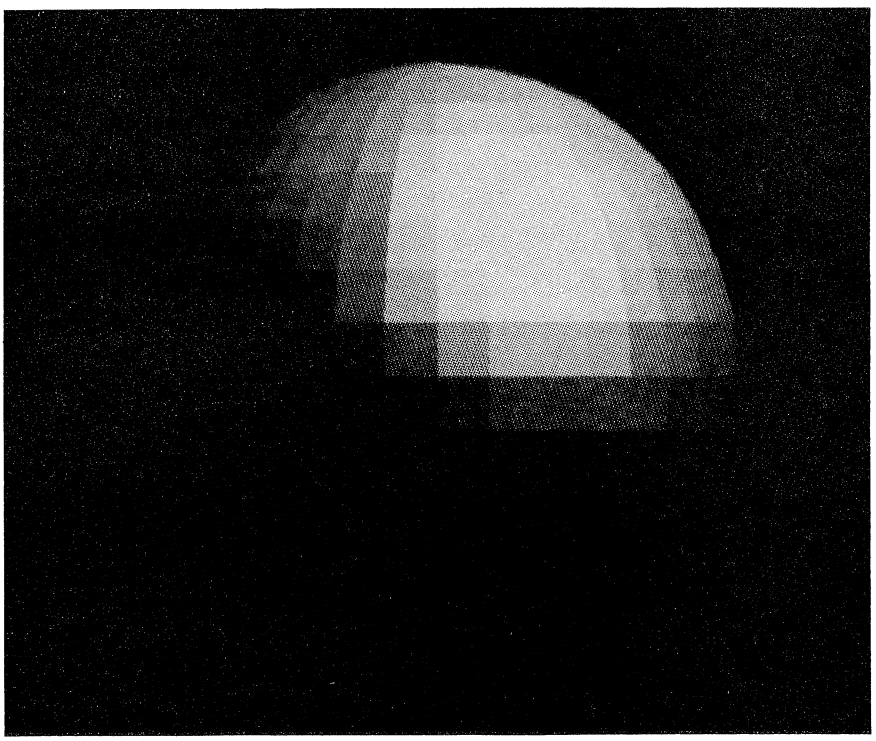
3684-17

Figure 5-11. Filling Panels: Replace and Overstrike Modes.

A. Not Filling Panel Boundaries.

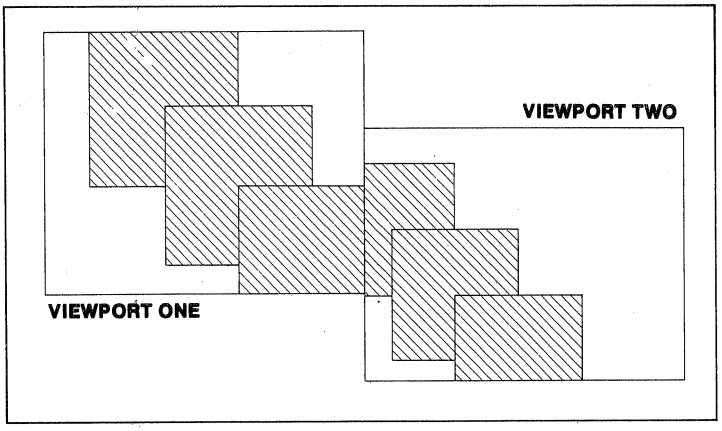


B. Filling Panel Boundaries.

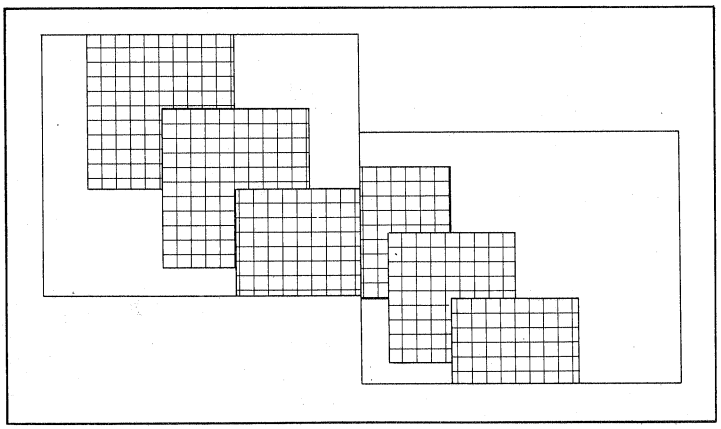


3684-18

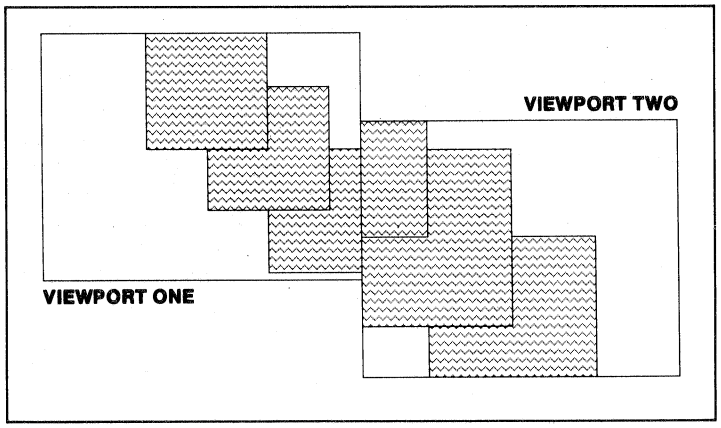
Figure 5-12. Effect of Filling Panel Boundaries.



Keying to the Current Viewport.



Keying to the Panel Being Filled.



Keying to the Screen.

3684-19B

Figure 5-13. Effect of Pattern Keying Mode.

DISPLAYING GRAPHTEXT

There are two ways of displaying text together with graphics on the terminal's screen:

- For compatibility with earlier TEKTRONIX terminals, the 4113 is capable of displaying alphatext together with graphics. However, to do this, the dialog area must be disabled (<enable-dialog-area: 0> command). (If the dialog area is enabled, all alphatext is directed to the dialog area.)
- A better way to display text together with graphics is by means of the <graphic-text> command. Text included in a <graphic-text> command is called *graphtext*. Graphtext is never displayed in the dialog area, regardless of whether the dialog area is enabled.

<GRAPHIC-TEXT> COMMAND

The <graphic-text> command has this syntax:

<graphic-text> = (ESC)(L)(T)
<string: *text-to-be-displayed*>

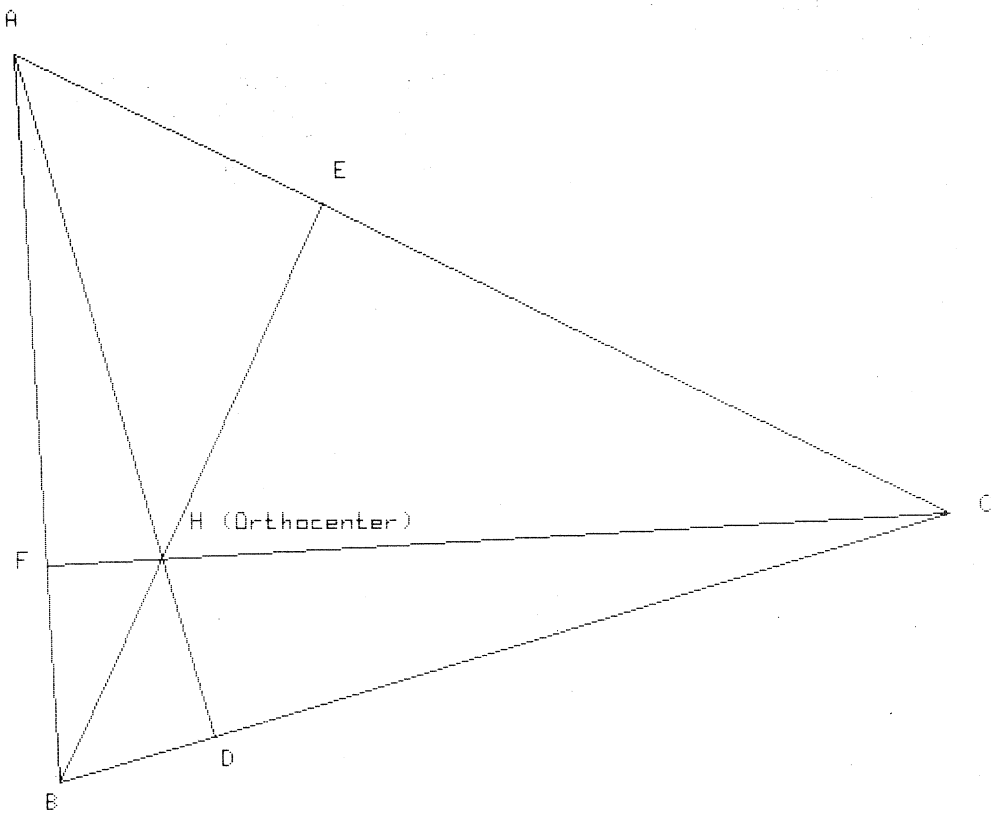
The text to be displayed is sent to the terminal as a <string> parameter. Since <string>s are <char-array>s, the only characters which may be displayed are those which are valid <char> parameters: the ASCII characters from (SP) to (~) — decimal equivalents from 32 to 126. Characters outside this range are ignored if they occur within a <string> parameter.

The first character of the text string is displayed with its lower left corner at the current graphic beam position. Figure 5-14 shows the effect.

The <graphic-text> command displays its graphtext with the lower left corner of the text string at the current graphic beam position. As each character of the string is displayed, the graphic beam position is moved to the lower left corner of the next character cell. At the end of the <graphic-text> command, the graphic beam position is just beyond the end of the graphtext string.

If displaying a character would move the graphic beam out of terminal space, that character (and the rest of the string) is not displayed, and the graphic beam position does not change.

```
<enter-vector-mode>
  <xy>
  <xy>
  .
  .
  .
<enter-vector-mode>
  <xy: (350,2912)>
  <graphic-text: "A">
<enter-vector-mode>
  <xy: (469,288)>
  <graphic-text: "B">
<enter-vector-mode>
  <xy: (3684,1285)>
  <graphic-text: "C">
.
.
.
<enter-vector-mode>
  <xy: (973,1231)>
  <graphic-text: "H (Orthocenter)">
```



3684-20

Figure 5-14. Effect of <Graphic-Text> Command.

<SET-GRAPHTEXT-SIZE> COMMAND

The <set-graphtext-size> command lets you set the width, height, and spacing of graphtext characters. The command has the following syntax:

<set-graphtext-size>
= (ESC)(M)(C)
  <int: *character-width*>
  <int: *character-height*>
  <int: *inter-character-spacing*>

The three <int> parameters are in terminal space units. The width and height must be in the range from 1 to 4095, while the spacing may range from 0 to 4095.

Figure 5-15 shows the effect of this command. In this figure, a <set-graphtext-size: 50,100,25> command was sent just before the <graphic-text: "H (Orthocenter)"> command.

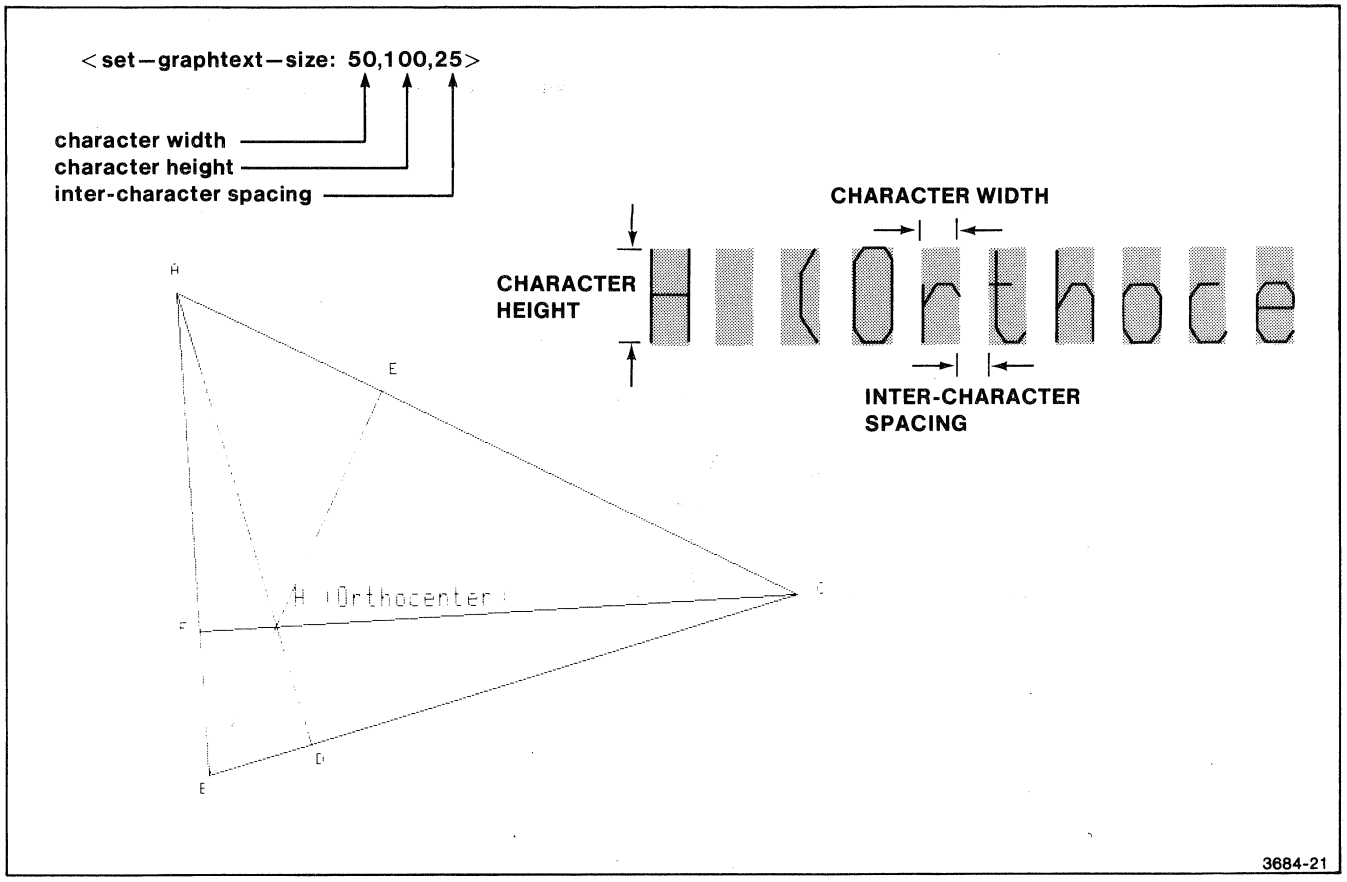


Figure 5-15. Effect of <Set-Graphtext-Size> Command.

<SET-GRAPHTEXT-ROTATION> COMMAND

The <set-graphtext-rotation> command lets you specify the counterclockwise rotation angle (in degrees) for all subsequent graphtext strings. (A negative rotation angle causes clockwise rotation.)

The command has the following syntax:

<set-graphtext-rotation>
= (ESC)(M)(R)
  <real: *angle-in-degrees*>

Figure 5-16 shows the effect. In the figure, a <set-graphtext-rotation: 30.0> comand was sent just before the <graphic-text: "H (Orthocenter)"> command.

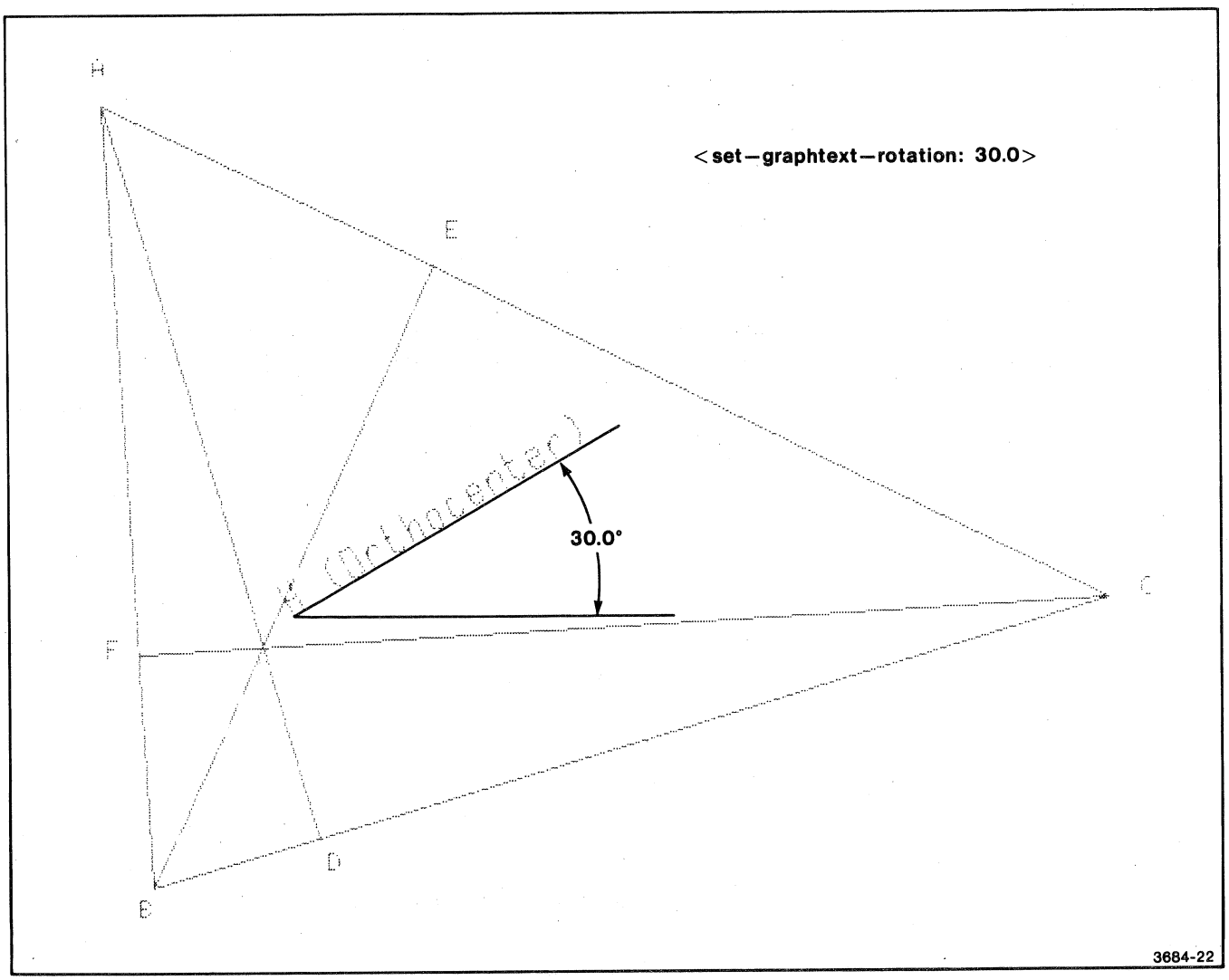


Figure 5-16. Effect of <Set-Graphtext-Rotation> Command.

<SET-GRAPHTEXT-PRECISION> COMMAND

The <set-graphtext-precision> command sets the precision level for subsequent graphtext. There are two precision levels: "stroke precision" (the default), and "string precision" (in which graphtext is drawn in the same way as alphatext). The command has the following syntax:

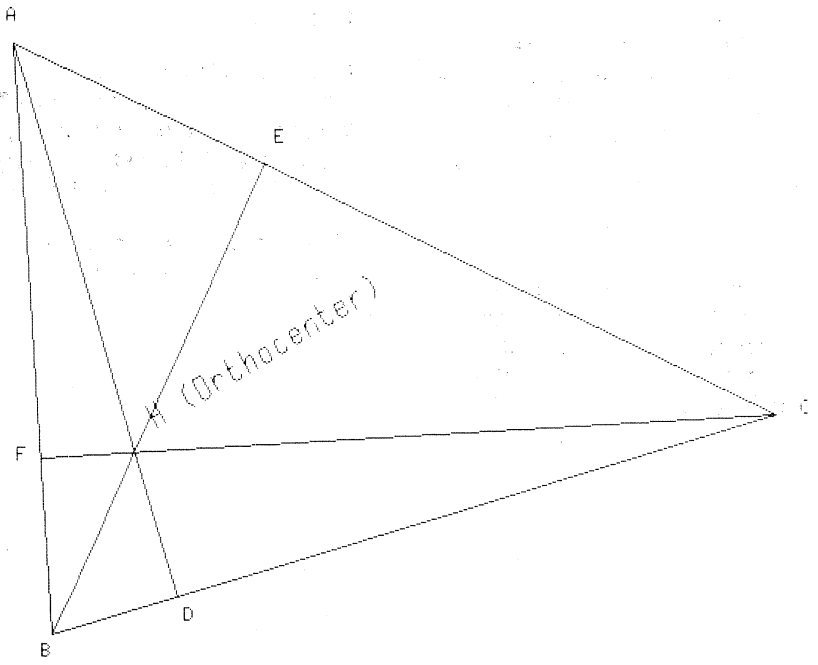
<set-graphtext-precision> = (ESC)(M)(Q)<int>

The <int> parameter is 2 for stroke precision, and 1 for string precision. Figure 5-17 shows the command's effect. String precision graphtext (and alphatext) cannot be rotated, nor can its size be changed with the <set-graphtext-size> command.

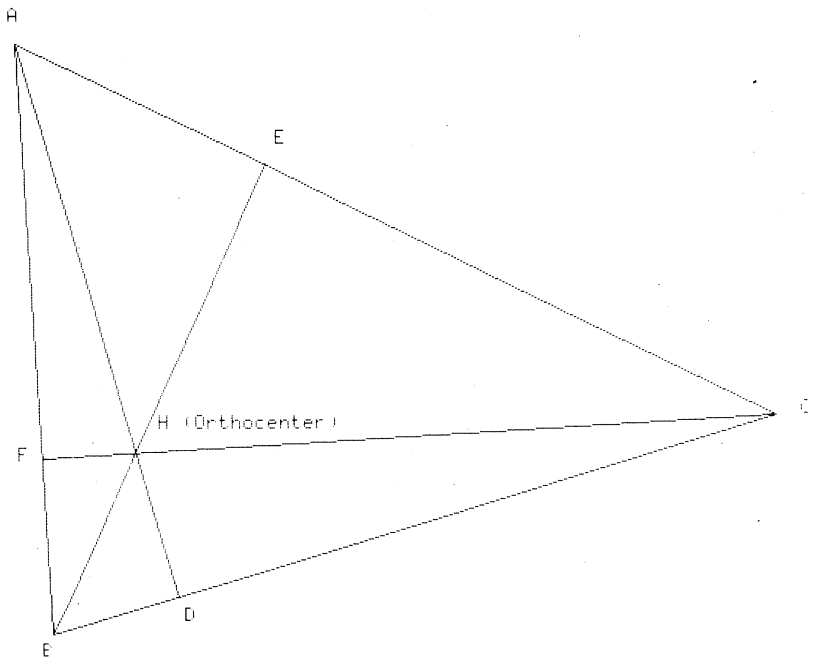
The <set-graphtext-font> command, described later in this section, does not affect string precision graphtext.

If you zoom in on a string precision graphtext (or on alphatext), the text does not appear larger. This is because alphatext and string precision graphtext are drawn directly in raster memory space, as dot patterns of pixels on the screen. Figure 5-18 shows the effect.

(See Section 7 for an explanation of "raster memory space," and of "zooming in.")



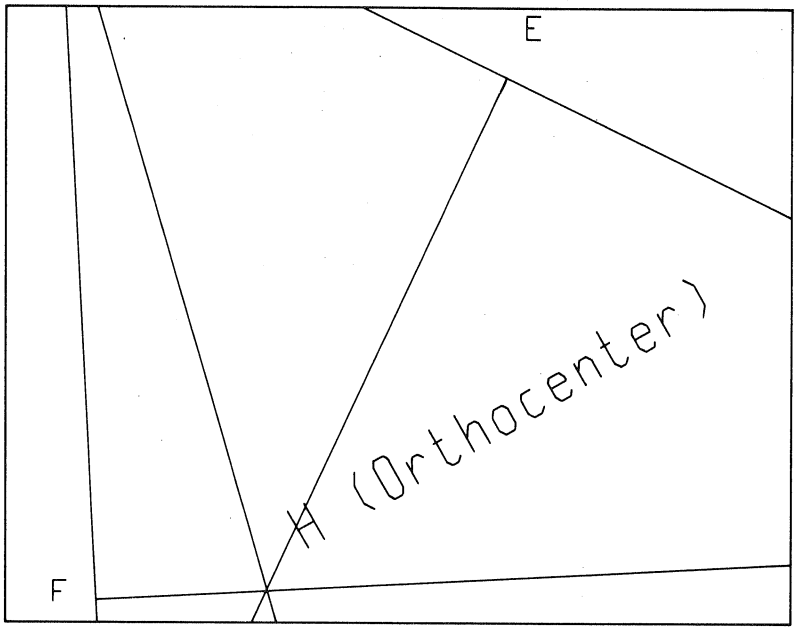
A. Stroke Precision Graphtext.



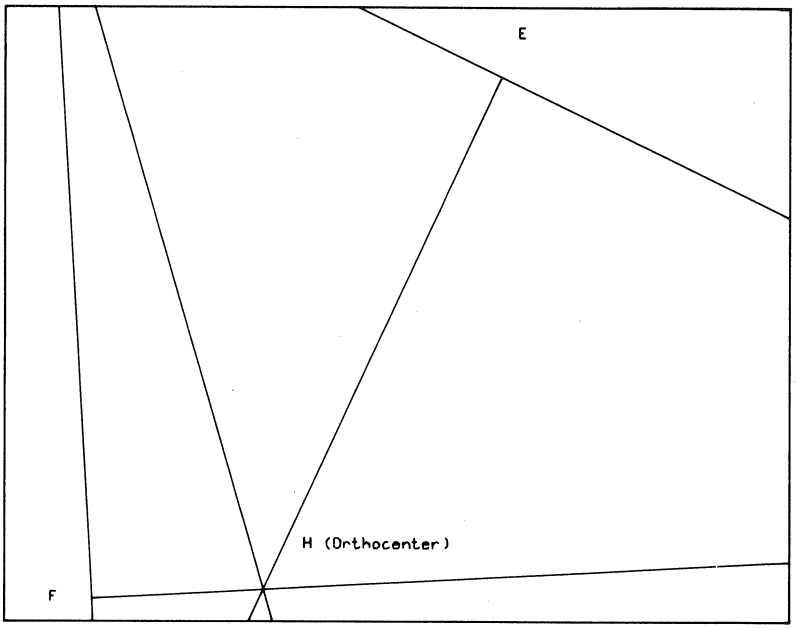
B. String Precision Graphtext.

3684-23

Figure 5-17. Effect of <Set-Graphtext-Precision> Command.



A. Stroke Precision Graphtext.



B. String Precision Graphtext.

3684-24

Figure 5-18. "Zooming In" On Graphtext.

PREDEFINED GRAPHTEXT FONTS

You can select alternate graphtext fonts with the <set-graphtext-font> command. When the terminal is turned on, there is always at least one predefined font: Font 0, the standard ASCII font. You can also create your own graphtext fonts. Such fonts are called user-defined fonts; how to create them is described later in this section.

Fonts Provided With Keyboard Options

If the terminal has an optional keyboard (Option 4A, 4C, 4E, or 4F) then Font 0 is not the only predefined graphtext font. Such a terminal also has predefined fonts 1, 3, 7, and 9. Table 5-2 lists these fonts. However, when the terminal is turned on, the default graphtext font is always the ASCII font, Font 0. This is true even for terminals equipped with optional keyboards. If you want to select another predefined font, you must do so explicitly with a <set-graphtext-font> command.

Figure 5-19. Fonts Provided With Keyboard Options.

Table 5-2

GRAPHTEXT FONTS SUPPLIED WITH OPTIONAL KEYBOARDS

Font Number	Graphtext Font
0	Standard ASCII font
1	Swedish font
3	United Kingdom font
7	APL font
9	Danish/Norwegian font

<SET-GRAPHTEXT-FONT> COMMAND

The <set-graphtext-font> command lets you select any of these fonts for displaying subsequent graphtext. The command has the following syntax:

<set-graphtext-font> = (ESC)(M)(F)
<int: *font-number*>

Figure 5-20 shows the effect of the <set-graphtext-font> command. In this figure, the font shown is a user-defined font. (Defining such a font is described later in this section.)

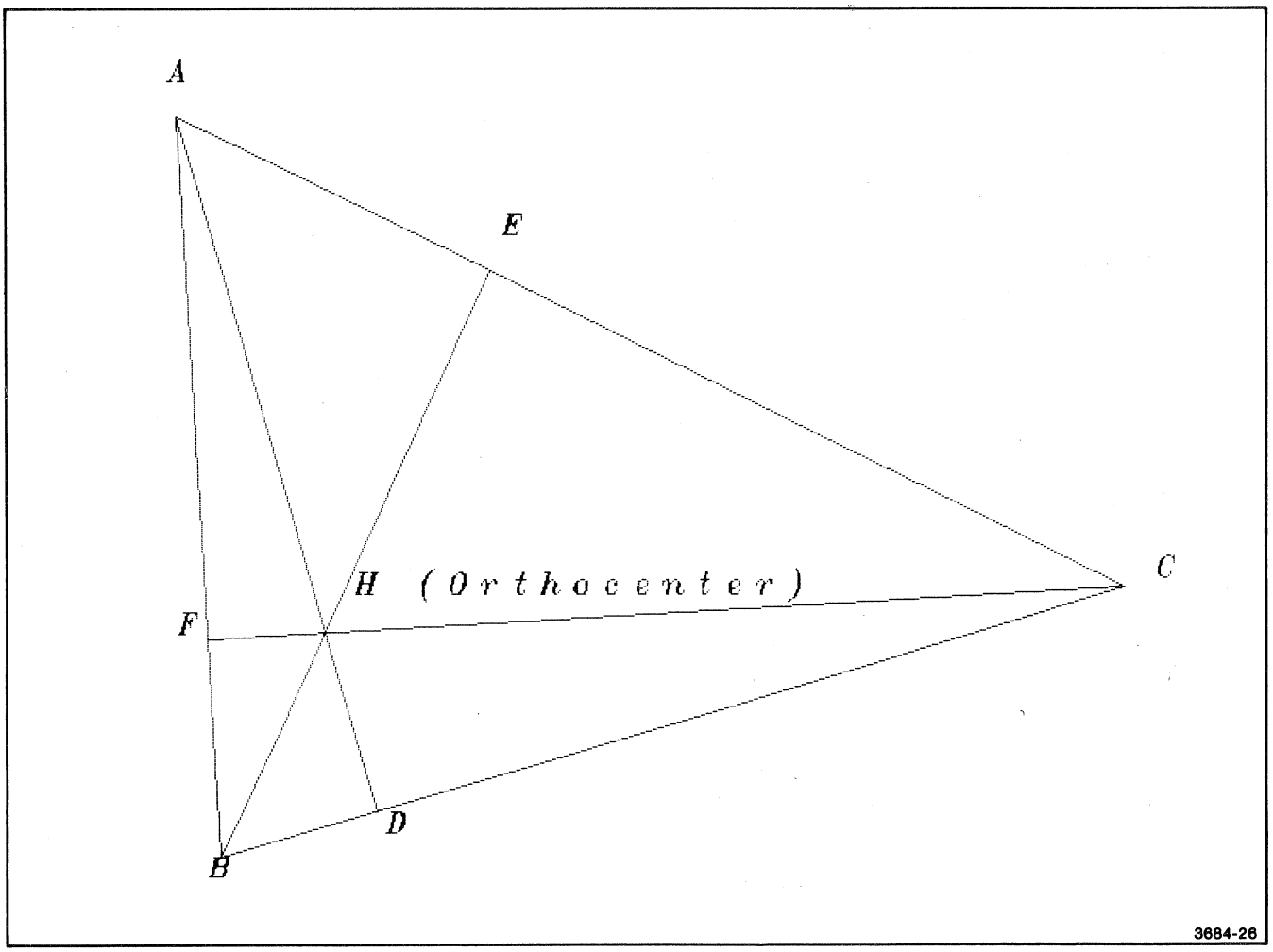


Figure 5-20. Effect of <Set-Graphtext-Font> Command.

DEFINING YOUR OWN GRAPHTEXT CHARACTERS

GENERAL PROCEDURE

The following steps show examples of commands for defining your own graphtext characters. These commands are described in more detail later in this section.

1. <Delete-graphtext-character: 2, −1>
   <Set-graphtext-font-grid: 2, 60, 80>

   Delete all user-defined characters in graphtext Font 2, and then specify a "font grid" for defining your own Font 2 graphtext characters.

2. <Set-pivot-point: (0,0)>

   The pivot point for subsequent graphtext characters will be at (0,0). When the character is displayed (by a <graphic-text> command), its pivot point appears at the current graphic beam position.

3. <Begin-graphtext-character: 2, 65>
   <enter-vector-mode>
   <xy>
   <xy>
   .
   .
   .
   <enter-vector-mode>
   <xy>
   <xy>
   .
   .
   .
   <end-graphtext-character>

   Define character number 65 (uppercase A) in graphtext Font 2. The character definition starts with the <begin-graphtext-character> command, and ends with the <end-graphtext-character> command. The character is defined as a series of vectors, or line segments. The vectors may be drawn with <enter-vector-mode> commands — (GS) characters — and <xy> parameters. Alternatively, they may be drawn with <move> and <draw> commands.

4. Repeat Steps 2 and 3 for each subsequent character definition. You can omit Step 2 (the <set-pivot-point> command) if the the character being defined is to have the same pivot point as the previous character defined.

<DELETE-GRAPHTEXT-CHARACTER> COMMAND

The <delete-graphtext-character> command has this syntax:

<delete-graphtext-character>
= (ESC)(S)(Z)
<int: *font-number*>
<int: *character-number*>

Font Number. The font number must be −1, or in the range from 0 to 32767. "Font −1" means "all user-defined fonts."

Character Number. The character number must be −1, or in the range from 32 to 126.

Specifying character number −1 deletes all user-defined characters in the specified font and also deletes the font grid for that font. If the font is not one of the predefined fonts (Font 0 in a standard terminal; Font 0, 1, 3, 7, or 9 in a terminal with a keyboard option), then specifying character number −1 also makes that font inaccessible. (The <set-graphtext-font> command results in an error if an attempt is made to select a font which is not predefined and does not have a font grid for user-defined characters.)

Specifying a character number from 32 to 126 deletes that particular user-defined graphtext character. The user-defined character is superceded by the corresponding predefined character. In most cases, this is the corresponding standard ASCII (Font 0) character. For Fonts 1, 3, 7, and 9, if an optional keyboard is installed, this is the corresponding character from the same predefined font.

<SET-GRAPHTEXT-FONT-GRID> COMMAND

The <set-graphtext-font-grid> command has the following syntax:

<set-graphtext-font-grid> = (ESC)(S)(G)
<int: *font-number*>
<int: *width-of-grid*>
<int: *height-of-grid*>

This command sets the width and height of a *graphtext font grid* used in defining graphtext characters. For each user-defined character in the specified font, the graphtext font grid is a rectangle extending above and to the right of that character's pivot point.

Suppose, for instance, that the width of the grid is 30, and the height is 40. If the uppercase A character (character number 65) is defined with its pivot point at (0,0), then its font grid extends from X= 0 to X= 30, and from Y= 0 to Y= 40. Likewise, if the lowercase g character is defined with its pivot point at (0,15), then its font grid extends from X= 0 to X= 30, and from Y= 15 to Y= 55. Figure 5-21 illustrates this.

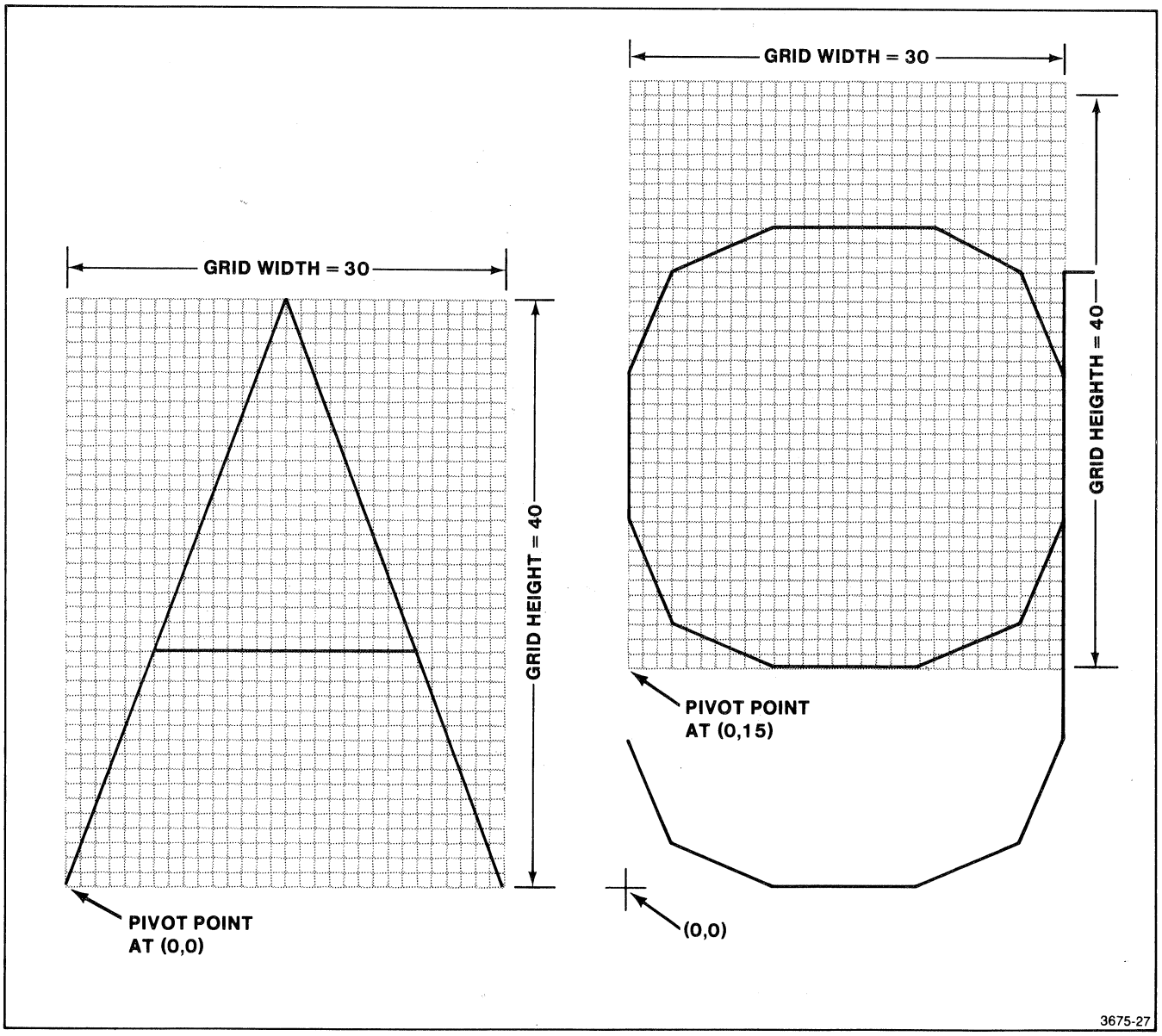


Figure 5-21. Font Grids for Two User-Defined Characters.

Later, when the character is displayed, the width and height of the font grid are mapped onto the current graphtext-size width and height, while the pivot point is mapped onto the current graphic beam position. Figure 5-22 illustrates this.

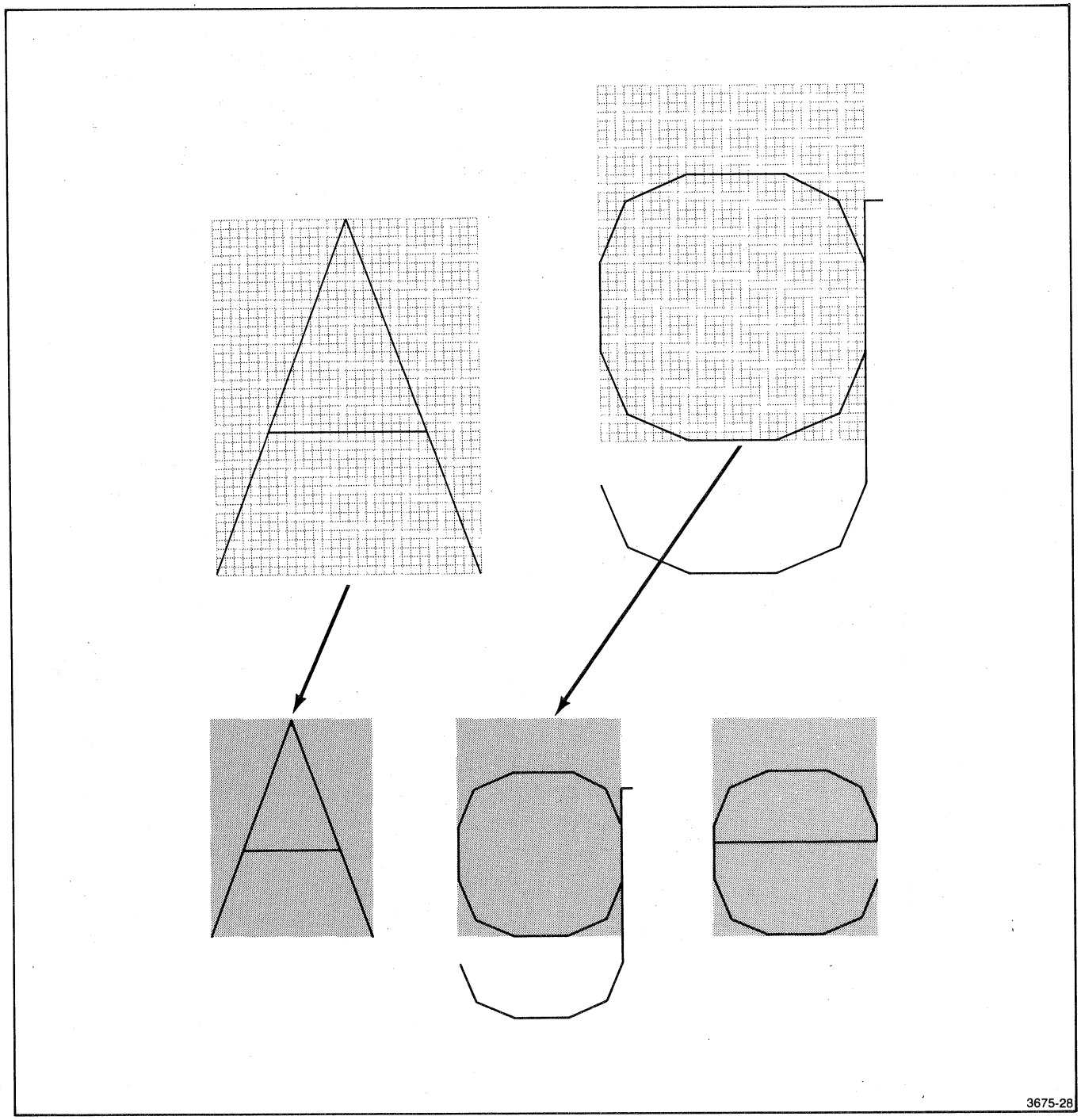


Figure 5-22. Displaying User-Defined Graphtext Characters.

All user-defined characters in a graphtext font share the same font grid. Therefore, issuing a <set-graphtext-font-grid> command is not allowed if there already exists a font grid for the specified font. (The terminal detects a type SG03 error; see the 4110 Series Command Reference Manual for details.)

Before sending a <set-graphtext-font-grid> command, therefore, you should delete any font grid which may already exist for that font. To do this, issue a <delete-graphtext-character> command with "minus one" as the character number; that not only deletes all the user-defined characters in the font, but also deletes the font grid.

In order to select a graphtext font which is not one of the predefined fonts, it is necessary that a font grid has been established for that font. For example, if no <set-graphtext-font-grid> command has been issued for Font 2, then the <set-graphtext-font: 2> command will result in an error.

<SET-PIVOT-POINT> COMMAND

The <set-pivot-point> command has this syntax:

<set-pivot-point> = (ESC)(S)(P)<xy>

It sets the pivot point for subsequent graphtext character definitions, as well as for subsequent segment definitions. (See Section 6 for a description of segment definitions.)

<BEGIN-GRAPHTEXT-CHARACTER> COMMAND

The <begin-graphtext-character> command has this syntax:

<begin-graphtext-character>
= (ESC)(S)(T)
  <int: *font-number*>
  <int: *character-number*>

This command starts the definition of a particular user-defined graphtext character. The font number must be in the range from 0 to 32767; the character number must be in the range from 32 to 126.

<END-GRAPHTEXT-CHARACTER> COMMAND

Each graphtext character definition ends with the <end-graphtext-character> command:

<end-graphtext-character> = (ESC)(S)(U)

Section 6

SEGMENTS

INTRODUCTION

DEFINITIONS

Segments (short for "picture segments") are graphic objects which can be manipulated by commands to the terminal. Each segment is referenced by its name, which is a number in the range from 1 to 32767.

For instance, suppose segment one has been defined to be a square with diagonals. Once segment one has been defined, you can issue commands to change its appearance and position. You can rotate and scale it about its "pivot point," move it from one position to another in terminal space, make it invisible ("turn it off"), cause it to "blink" on and off, and even use it as the graphic cursor for GIN (graphic input) operations.

Each segment has a *pivot point*, which is set at the time the segment is created. It is the pivot point within a segment which serves to define the segment's position. Thus, a <set-segment-position> command to position segment one at the point (100,500) actually positions only the pivot point for that segment at the point (100,500).

The pivot point is also the point about which scaling and rotation occurs. These operations are described later, under "Scaling and Rotation."

CREATING A SEGMENT

The <begin-segment> and <end-segment> commands mark the beginning and end of a segment definition. The syntax for these commands is as follows:

<begin-segment> = (ESC)(S)(O)
<int: *segment-number*>

<end-segment> = (ESC)(S)(C)

Figure 6-1 shows commands which create a simple segment. (All these commands are described in more detail in the 4110 Series Command Reference Manual.)

This segment, segment one, is a square with diagonals, 200 units on a side. Its pivot point is at (100,100), the center of the square, where the diagonals intersect.

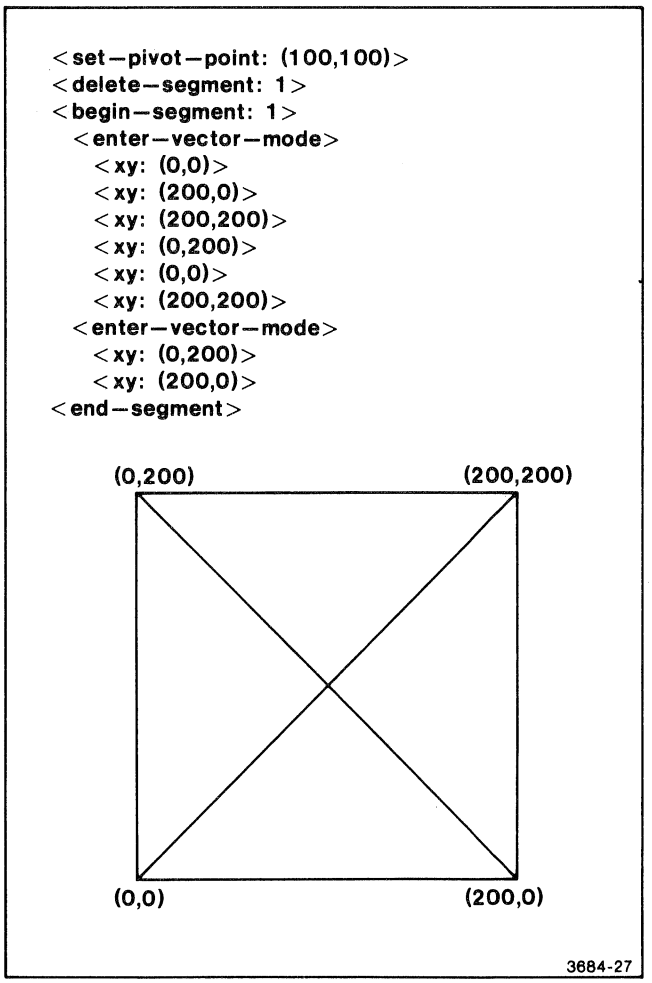


Figure 6-1. Segment One, A Square With Diagonals.

In Figure 6-1, the <set-pivot-point> command is issued *before* the <begin-segment> command. This is necessary because, once a segment has been defined, its pivot point cannot be changed. A <set-pivot-point> command, therefore, affects only the pivot points of segments which are defined *after* that <set-pivot-point> command.

The <delete-segment> command deletes any "segment one" which might already exist. This is necessary because one cannot create a new segment with a segment number equal to that of an existing segment.

If segment one does not exist, and the terminal's error threshold is set to one or less, then the <delete-segment> command causes the terminal to display a type SK10 warning message: "Existence Problem in Parameter 1 of (ESC)(S)(K) Command." You can suppress this message by setting the error threshold to 2 or more. For details, see the description of the <set-error-threshold> command in the 4110 Series Command Reference Manual; see also Appendix C, "Error Codes," in that manual.

The <begin-segment> and <end-segment> commands begin and end the segment definition. Any commands which occur between those two commands are included in the definition of the segment. Figure 6-1 has <enter-vector-mode> commands and <xy> coordinates which perform moves and draws to create the square and its two diagonals.

The <begin-higher-segment> command provides an alternate method for creating segments. This command closes the segment currently being defined and begins a new segment with a segment number that is one greater than the segment just closed. The pivot point and position of the new segment is set to the current beam position. The <begin-higher-segment> command does the work of several commands. Note that a segment must be in the process of being defined when this command is issued. The current pivot point and default segment position are not changed by this command.

The <begin-lower-segment> command works like the <begin-higher-segment> command except that segment number decreases by one, rather than increases.

The <begin-new-segment> command also works similarly to the <begin-higher-segment> command. Rather than incrementing the segment number by one, <begin-new-segment> lets you specify a segment number for the new segment.

See the 4110 Series Command Reference Manual for detailed descriptions of the <set-pivot-point>, <begin-segment>, <end-segment>, and <delete-segment> commands. There you will find the details of the command syntax, which you can use to write subroutines to issue these commands.

INCLUDING OTHER SEGMENTS IN A SEGMENT DEFINITION

When defining a segment, you can include a copy of another segment in the segment definition. To do this, use the <include-copy-of-segment> command:

<include-copy-of-segment>
= (ESC)(L)(K)<int: *segment-number*>

This command causes an image of the specified segment, in its current position, to be included as part of the segment being defined.

Figure 6-2 illustrates the process. In the figure, <set-segment-position> and <set-segment-image-transform> commands are used to reposition segment one before including copies of segment one in the definition of segment two. The <set-segment-position> and <set-segment-image-transform> commands are described later in this section.

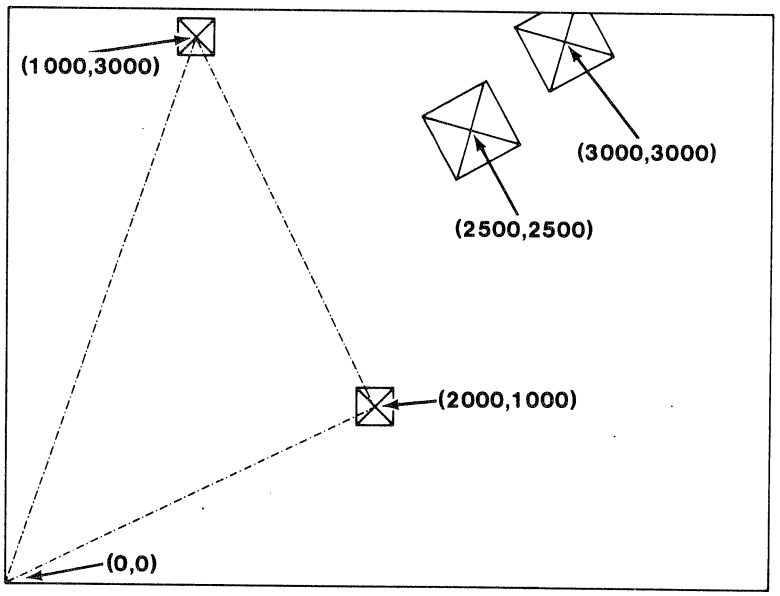
RETAINED AND NON-RETAINED SEGMENTS

In the computer graphics literature, you may find the terms *retained segment* and *non-retained segment*. Retained segments are segments which are retained in memory (by the host computer or by the terminal) after they have been drawn. All the 4113's segments are retained segments.

The so-called non-retained segment consists of all graphics which are drawn but are not retained. For the 4113, this is everything *not* included within the definition of a numbered segment.

Graphics not stored in segments (that is, graphics which comprise the so-called non-retained segment) are lost whenever a <page> command (or the PAGE key) erases the current viewport. (For now, regard "current viewport" as meaning the same thing as "the terminal's screen." For information on viewports, see Section 7.)

```
<begin-segment: 2>
  <set-line-style: 2>
  <enter-vector-mode>
    <xy: (0,0)>
    <xy: (2000,1000)>
    <xy: (1000,3000)>
    <xy: (0,0)>
  <set-segment-position: 1,(2000,1000)>
  <include-copy-of-segment: 1>
  <set-segment-position: 1,(1000,3000)>
  <include-copy-of-segment: 1>
  <set-segment-image-transform: 1,2.0,2.0,30.0,(3000,3000)>
  <include-copy-of-segment: 1>
  <set-segment-position: 1,(2500,2500)>
  <include-copy-of-segment: 1>
<end-segment>
```



3684-28

Figure 6-2. Including One Segment in the Definition of Another.

Graphics stored in segments, however, are not lost when a <page> occurs. Instead, after erasing the current viewport, the 4113 redraws all graphics stored in visible segments. (If you want a segment to disappear when you erase the viewport, you can make it invisible with the <set-segment-visibility> command, described later in this section. Or, you can delete the segment — issue a <delete-segment> command.)

STATIC AND DYNAMIC ATTRIBUTES

Those attributes, or properties, of a segment which cannot be changed once the segment has been defined are called *static attributes* of the segment. Those attributes which can be changed after segment definition are called *dynamic segment attributes*.

STATIC ATTRIBUTES

PIVOT POINT

There is just one static segment attribute: the pivot point. Once a segment has been defined, its pivot point cannot be changed. (To set a segment's pivot point, you must issue the <set-pivot-point> command *before* defining the segment. A <set-pivot-point> command issued after the segment is defined affects only segments defined later.)

The pivot point is important when moving a segment: changing its position in terminal space, rotating it, or changing its size. A segment's "position" is defined by the position of its pivot point. Also, the pivot point is the only point of the segment which does not move when the segment is scaled in the x- and y-directions or rotated. These operations are described later in this section, under "Position" and "Scaling and Rotation." They are also described in the Command Reference Manual, under the <set-segment-position> and <set-segment-image-transform> commands.

PRIMITIVE ATTRIBUTES

A segment's pivot point, however, is not the only thing about a segment which is static. All graphic primitives within a segment (lines, graphtext font, graphtext size, alphatext font, etc.) are static, unchangeable parts of the segment. That is, once a segment has been created, it cannot be edited; it can only be deleted and defined again.

Any attributes of the graphic primitives within a segment are also static and unchangeable. For instance, the lines within a segment are graphic primitives; the line styles used to draw those lines are attributes of those lines. Thus, if a segment is defined with its lines drawn in line style two (dashed lines), then it will always be drawn with its lines in that style.

Again, graphtext within a segment is deemed to be a graphic primitive. The graphtext font and graphtext size are attributes of the graphtext graphic primitive. As primitive attributes, they are static, unchangeable parts of the segment. Thus, if a segment is defined with graphtext in a certain graphtext font and a certain size, then it will always be drawn with that graphtext in that font and that style.

Table 6-1 lists the graphic primitives and their primitive attributes, together with the commands which embed those primitives and primitive attributes in a segment. Once a segment has been defined, the primitives and primitive attributes in Table 6-1 are static, unchangeable parts of the segment.

Table 6-1

GRAPHIC PRIMITIVES AND PRIMITIVE ATTRIBUTES

Graphic Primitives	Primitive Attributes	Related Commands
Lines	— —	<Enter-vector-mode>, <xy>, <Move>, <Draw>
	Line Style	<Set-line-style>
	Line Index	<Set-line-index>
	Line Width	<Set-line-width>
Graphtext	— —	<Graphic-text>
	Graphtext size	<Set-graphtext-size>
	Graphtext font	<Set-graphtext-font>
	Graphtext precision	<Set-graphtext-precision>
	Graphtext rotation	<Set-graphtext-rotation>
	Text index	<Set-text-index>
Alphatext	— —	<Enter-alpha-mode>
	Alphatext font	<Set-alphatext-font>
	Text index	<Set-text-index>
Markers	— —	<Enter-marker-mode>, <xy>
	Marker type	<Set-marker-type>
	Line index	<Set-line-index>
Panels	— —	<Begin-panel-boundary>, <End-panel>
	Fill pattern	<Select-fill-pattern>
	Panel filling mode	<Set-panel-filling-mode>

Information about the commands in Table 6-1 can be found, in alphabetical order, in the 4110 Series Command Reference Manual. Descriptions of the commands can also be found in various sections of this manual:

Section 4: <set-alphatext-font>

Section 5: <enter-vector-mode>
<move>
<draw>
<set-line-style>
<begin-panel-boundary>
<end-panel>
<select-fill-pattern>
<set-panel-filling-mode>

Section 7: <set-line-index>
<set-text-index>

DYNAMIC SEGMENT ATTRIBUTES

The *dynamic attributes* of a segment are those attributes, or qualities, which can be changed after the segment is defined. These attributes are: position, scale factors, rotation, visibility, writing mode, highlighting, detectability, display priority, and segment class.

POSITION

When a segment is defined, its "position" is the same location in the terminal's 4096-by-4096 coordinate space as its pivot point. In the example earlier in this section, segment one was a square with diagonals, defined with the pivot point at the center of the square — the point (100,100). Thus, the initial position of segment one is also the point (100,100).

The <set-segment-position> command moves the segment so that its pivot point is displayed at some new position.

What happens to the old image of the segment? That depends on the current "fixup level," as determined by the most recent <set-fixup-level> command. If the fixup level is four or higher, then the old image is removed from view. If the fixup level is less than four, the old image is not removed until the next <renew-view> or <page> command. (For details, see the <set-fixup-level> command description in the 4110 Series Command Reference Manual.)

The <set-segment-position> command has the following syntax:

<set-segment-position> = (ESC)(S)(X)<xy>

Here, the <xy> parameter names the point in terminal space where the segment's pivot point is to be displayed.

Figure 6-3 demonstrates the <set-segment-position> command. In Part A of the figure, the segment is defined with the pivot point at (100,100). In Part B, the segment is moved so that the pivot point is displayed at (1000,1000); then the screen is erased (<page> command) in order to eliminate the old image.

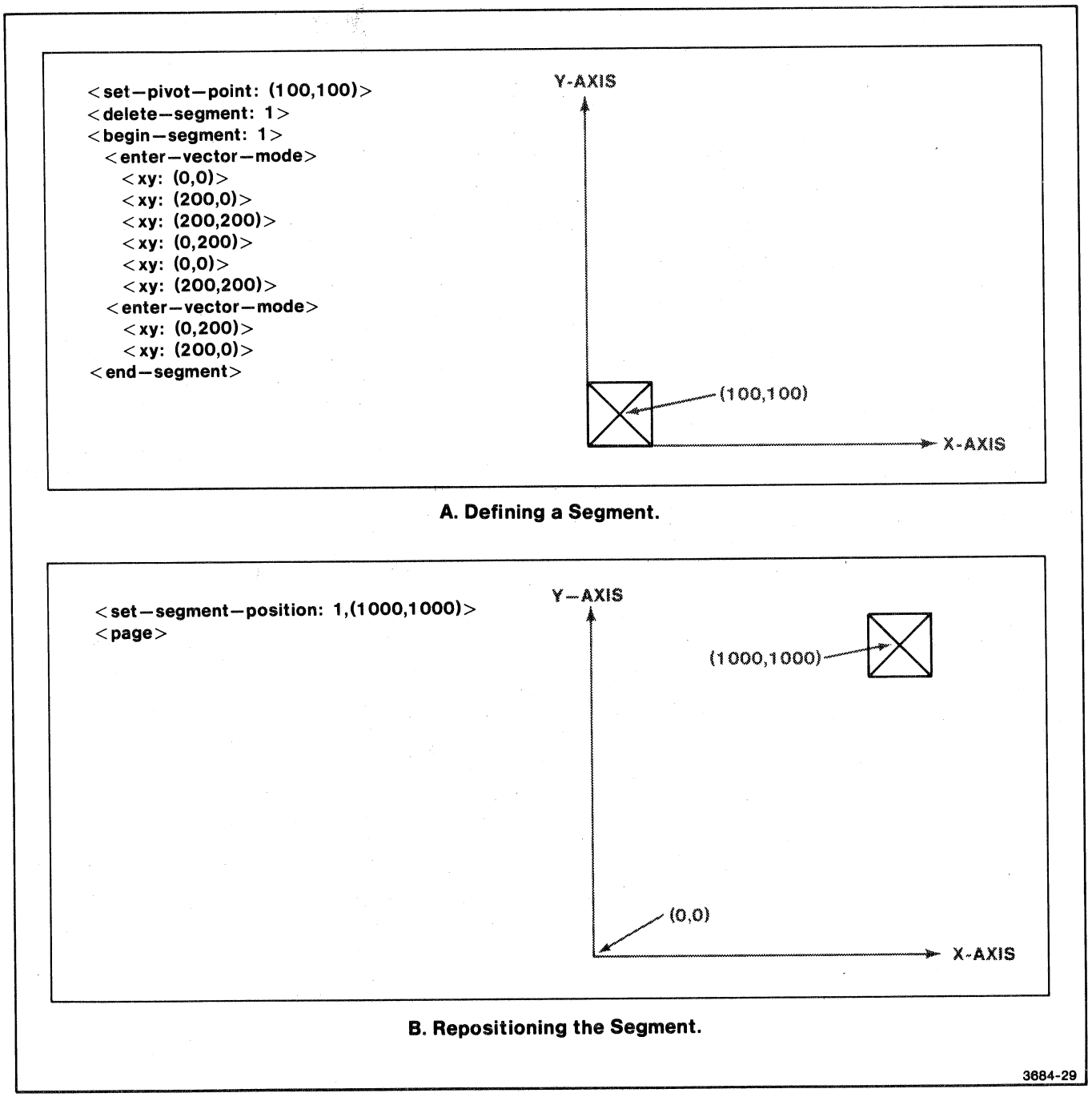


Figure 6-3. Effect of the <Set-Segment-Position> Command.

The next figure shows what happens when you move a segment to the edge of the 4113's 4096-by-4096 terminal space. That part of the segment which falls outside the current window is not displayed.

(The term "current window" is defined in Section 7. For this example, the window is assumed to have its initial value on power-up: from X= 0 to X= 4095, and from Y= 0 to Y= 3127. See Section 7 for more information on windows and viewports.)

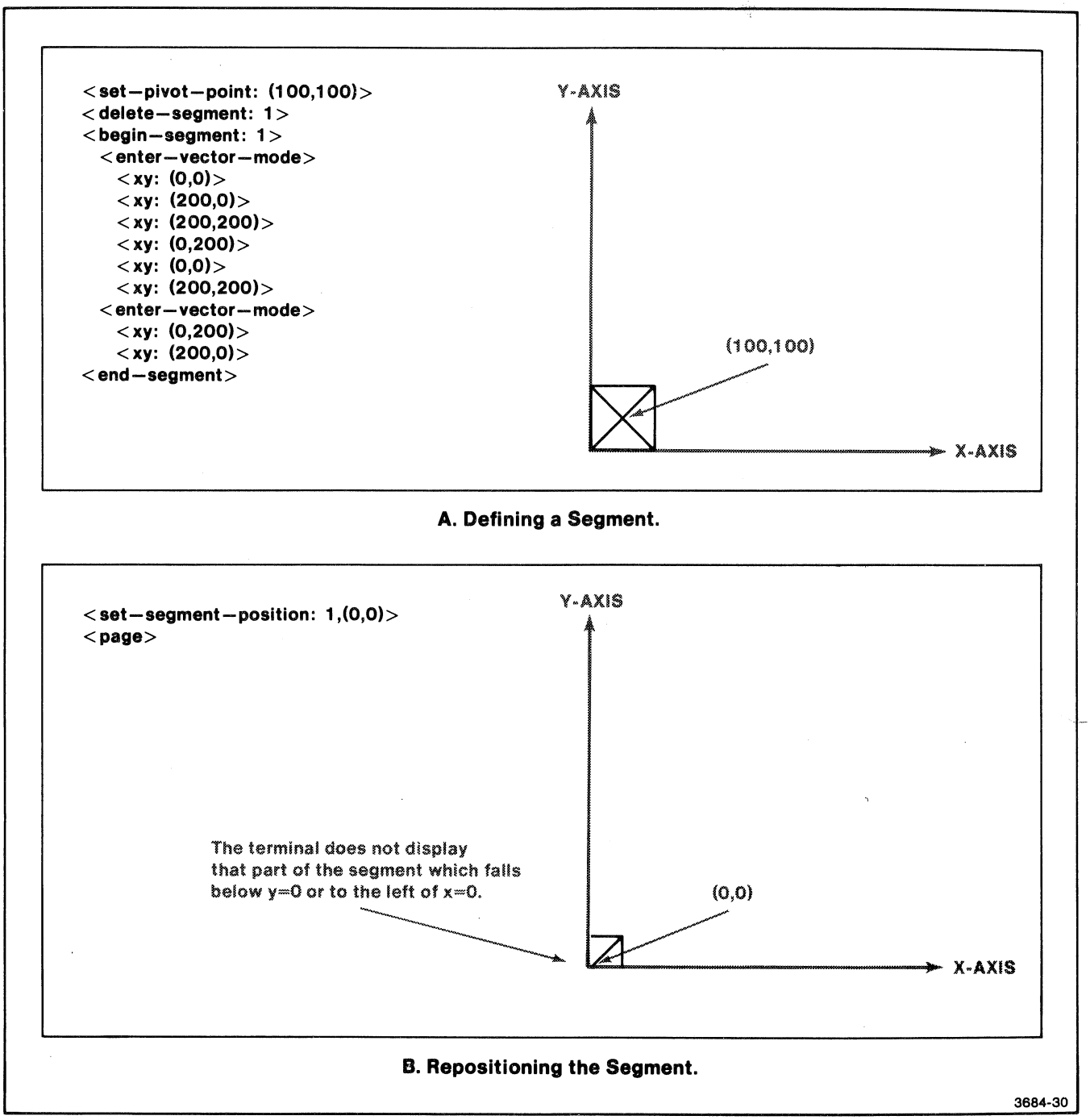


Figure 6-4. Positioning a Segment at the Edge of the Screen.

SCALING AND ROTATION

The <set-segment-image-transform> command performs three operations on a segment: scaling, rotation, and positioning. The syntax of this command is as follows:

<set-segment-image-transform>
= (ESC)(S)(I) <int><real><real><real><xy>

Here, the first parameter is an <int> specifying the segment number, the second and third parameters are the x- and y-scaling factors, the fourth parameter is the rotation angle in degrees, and the fifth parameter specifies the point at which the segment is positioned after the scaling and rotation has been performed.

First, the segment is scaled in the x- and y-directions about its pivot point. That is, the pivot point is the invariant point for the scaling operation — the only point which does not "move" because of the x- and y-scaling.

Second, the segment is rotated about its pivot point by the number of degrees given in the "rotation" parameter.

Finally, the segment is positioned (translated) so that its pivot point is displayed in the location specified by the final <xy> parameter. (If you only want to change a segment's position without affecting the other image transform parameters, use the <set-segment-position> command rather than the <set-segment-image-transform> command.)

All three operations begin with the segment as it was originally defined. That is, they are "absolute" rather than "relative" operations.

Examples

Figure 6-5 shows commands which do the following, and the effects of those commands:

1. Define a segment: segment one, a square of side 200 with diagonals, pivot point at (100,100).
2. Scale segment one in the x-direction by a factor of three, and position it at its original position: (100,100). Then page the screen to remove the old image of the segment.
3. Scale segment one in the y-direction by a factor of three, move it to (1000,1000), and then page the screen to remove the old image.

Figure 6-6 shows more examples, continuing from those of Figure 6-5:

4. Scale segment one in the x-direction by a factor of 2.0 and in the y-direction by a factor of 3.0; rotate it counterclockwise by 30 degrees; position it at (1000,1000). Then erase the screen to remove the old image.
5. Scale segment one in the x-direction by a factor of 0.0 and in the y-direction by a factor of 4.0; rotate it clockwise by 30 degrees (counter-clockwise by −30 degrees); position it at (1000,1000). Then erase the screen to remove the old image.

VISIBILITY

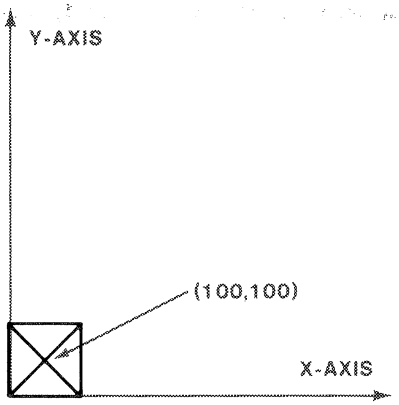
You can make a segment visible or invisible with the <set-segment-visibility> command:

<set-segment-visibility> = (ESC)(S)(V)<int><int>

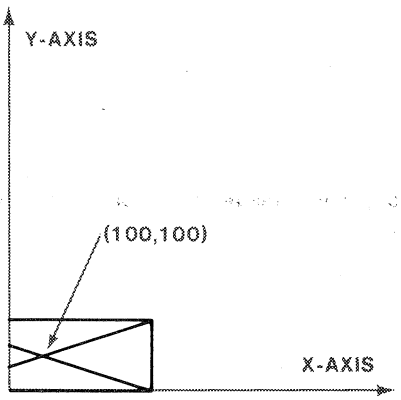
Here, the first <int> is the segment number. The second <int> is one to make the segment visible, or zero to make the segment invisible. An invisible segment is retained in memory but is not redisplayed when the screen is erased.

For more details, see the description of the <set-segment-visibility> command in the 4110 Series Command Reference Manual.

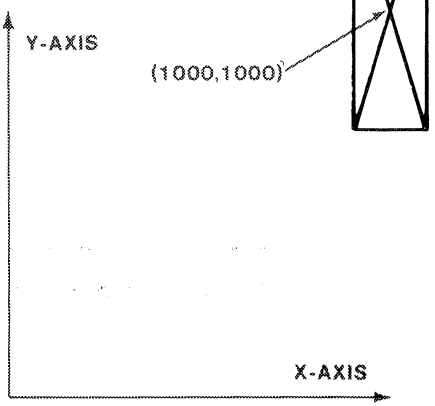
```
<set-pivot-point: (100,100)>
<delete-segment: 1>
<begin-segment: 1>
  <enter-vector-mode>
    <xy: (0,0)>
    <xy: (200,0)>
    <xy: (200,200)>
    <xy: (0,200)>
    <xy: (0,0)>
    <xy: (200,200)>
  <enter-vector-mode>
    <xy: (0,200)>
    <xy: (200,0)>
<end-segment>
```




```
<set-segment-image-transform: 1,3.0,1.0,0.0,(100,100)>
<page>
```




```
<set-segment-image-transform: 1,1.0,3.0,0.0,(1000,1000)>
<page>
```



3684-31

Figure 6-5. Examples of <Set-Segment-Image-Transform> Command.

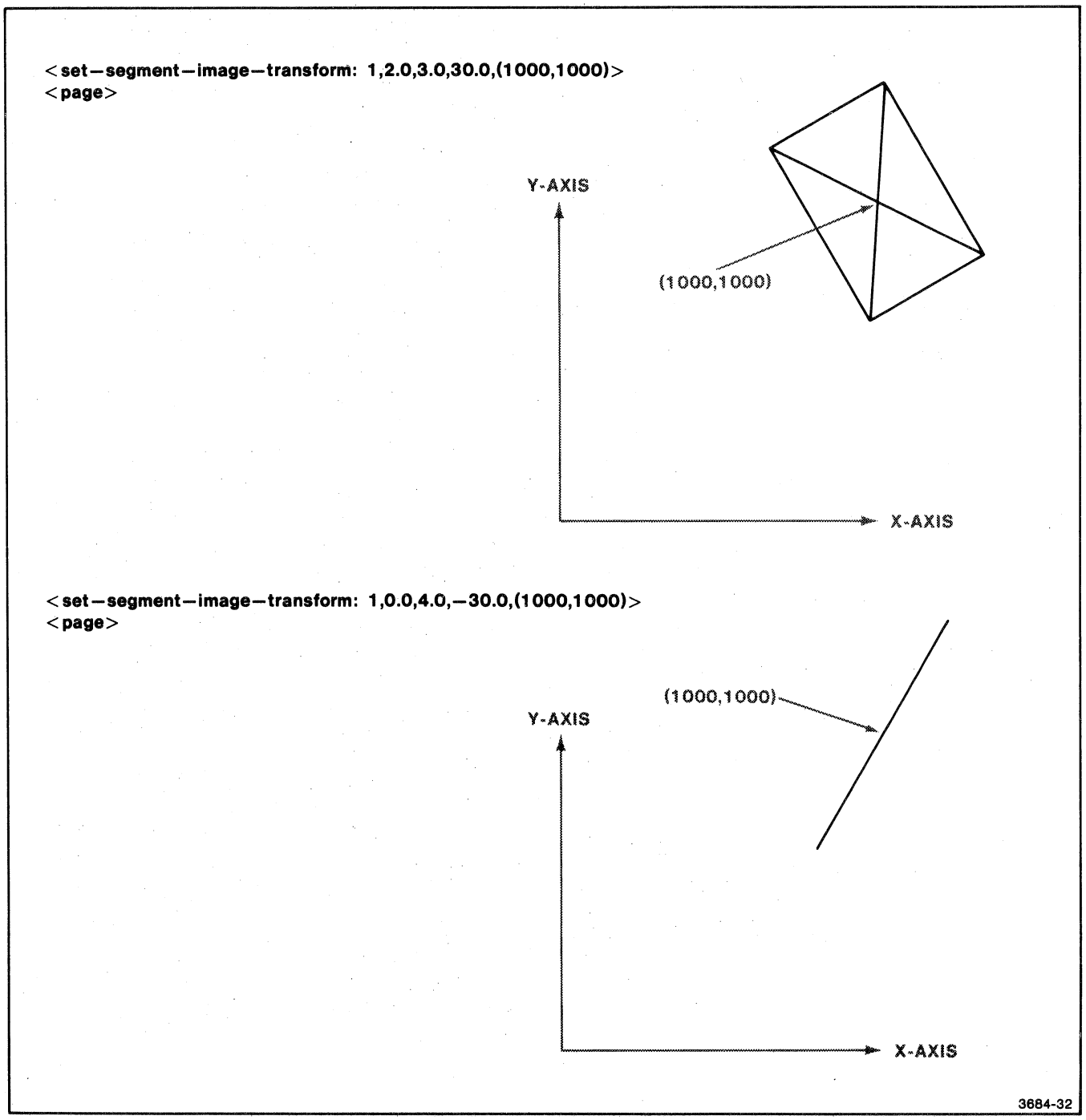


Figure 6-6. More Examples of <Set-Segment-Image-Transform>.

WRITING MODE

The <set-segment-writing-mode> command lets you control how, when a segment is displayed, it is written in the raster memory buffer. (The raster memory buffer is described in Section 7.) There are two modes: *set mode* and *XOR mode*.

In set mode, each pixel in the raster memory buffer which is changed is replaced with a pixel in the color-index being written to the raster memory buffer. (For instance, in drawing a line, whenever a pixel is changed, it is replaced with a new pixel in the current line index.)

In XOR mode, however, when a pixel in the raster memory buffer is changed, it is replaced with a pixel whose color-index is the result of a bit-by-bit exclusive-or operation. For instance, when drawing a line, the old color-index (of each pixel to be changed) is combined, bit by bit, in an exclusive-or operation with the current line index. The result of the exclusive-or operation gives the color-index which is written into the raster memory buffer.

Unlike lines drawn in set mode, lines drawn in XOR mode can be removed without leaving a trace. (To do so, you redraw the line again in XOR mode.) However, where two lines drawn in XOR mode cross each other, the intersection point may be displayed in a color-index different from the color-index of either line.

If you want a segment's lines to "look correct," even at points of intersection, you should display the segment in set mode. If you want to be able to remove the segment or move it about without leaving any traces of its former presence, you should display the segment in XOR mode.

The <set-segment-writing-mode> command has this syntax:

<set-segment-writing-mode>
= (ESC)(S)(M)<int><int>

The first <int> parameter is the segment number. The second <int> specifies the writing mode: one for set mode, two for XOR mode.

For more details, see Section 7 and the description in the 4110 Series Command Reference Manual of the <set-segment-writing-mode> command.

HIGHLIGHTING

You can draw the operator's attention to a segment by causing it to "blink" (become invisible, then become visible again).

The <set-segment-highlighting> command has this syntax:

<set-segment-highlighting>
= (ESC)(S)(H)<int><int>

Here, the first <int> is the segment number. The second <int> is one if the segment is to be highlighted, zero if the highlighting feature is to be turned off.

For more details, see the description of the <set-segment-highlighting> command in the 4110 Series Command Reference Manual.

DETECTABILITY

A segment's "detectability" attribute determines whether or not the segment can be "picked" during a graphic input "pick" operation. For more details, see Section 8, which describes graphic input; see also the descriptions of the <enable-GIN> and <set-segment-detectability> commands in the 4110 Series Command Reference Manual.

The <set-segment-detectability> command has this syntax:

<set-segment-detectability>
= (ESC)(S)(D)<int><int>

The first <int> parameter is the segment number. The second <int> is one if the segment is to detectable (pickable), and zero if it is not to be detectable.

DISPLAY PRIORITY

A segment's "display priority" is a number which determines the order in which segments are redrawn when the screen is erased and the order in which segments are traversed during a graphic input "pick" operation. The display priority may be any integer in the range from −32768 to + 32767. On power-up, the default display priority for new segments is zero.

When segments are redrawn after the screen is erased, segments with high display priority are drawn after those with lower priority. That way, higher-priority segments over-write lower-priority ones.

During a graphic input pick operation, the highest-priority segments are examined first. That way, if parts of several segments fall within the pick aperture, the highest-priority segment is the one which is picked.

The <set-segment-display-priority> command has this syntax:

<set-segment-display-priority>
= (ESC)(S)(S)<int><int>

The first <int> parameter is the segment number. The second parameter is the display priority.

SPECIAL SEGMENT NUMBERS

All the segments which you define (with the <begin-segment> and <end-segment> commands) must have segment numbers in the range from 1 to 32767. However, in the commands which change the dynamic attributes of segments, you can use four other special segment numbers: 0, −1, −2, and −3.

SEGMENT ZERO

"Segment Zero" is the crosshair graphics cursor. The crosshair cursor is generated by special circuitry within the terminal, and you cannot manipulate it in all the ways that other segments can be manipulated. You can move the crosshair cursor by specifying segment zero in the <set-segment-position> command. However, you cannot rotate or scale the crosshair cursor, so "segment zero" is not allowed as a parameter in the <set-segment-image-transform> command. To see whether segment number zero is allowed for a particular command, refer to that command's description in the 4110 Series Command Reference Manual.

SEGMENT MINUS ONE

"Segment Minus One" means "all segments currently defined." (That is, all segments with numbers from 1 to 32767; segment zero is not included.) For instance, you can make all segments invisible by specifying segment minus one in a <set-segment-visibility> command:

<set-segment-visibility: −1, 0>

Likewise, you can position all segments at a given point, force all segments to be displayed in storage mode, and highlight all segments:

<set-segment-position: −1, (1000,1000)>
<set-segment-writing-mode: −1, 1>
<set-segment-highlighting: −1, 1>

No doubt other examples will occur to you. To see whether "segment minus one" is allowed as a parameter for a particular command, see that command's description in the 4110 Series Command Reference Manual.

SEGMENT MINUS TWO

"Segment Minus Two" means "the default for all segments not yet defined." That is, if you want to control a segment's visibility, or writing mode, or whatever, *even before that segment is created*, you can use a command specifying segment minus two *before* the <begin-segment> command which starts the definition of that segment.

For example, the following commands cause a segment to be defined and to be displayed only *after* the segment definition is complete:

```
<set-segment-visibility: −2, 0>
<begin-segment: 1>
   <enter-vector-mode>
      <xy>
      <xy>
      .
      .
      .
   <enter-vector-mode>
      <xy>
      <xy>
      .
      .
      .
<end-segment>
<set-segment-visibility: 1, 1>
```

Using "segment minus two," you can specify default values (for new segments) of the following dynamic segment attributes: detectability, display priority, highlighting, visibility, position, and writing mode. (You can also set the "segment classes" to which new segments belong; this section discusses "segment classes" later in more detail.)

SEGMENT MINUS THREE

"Segment Minus Three" means "all segments in the current segment matching class."

To understand "current segment matching class," you must understand the concept of "segment classes," which is the next topic in this section.

SEGMENT CLASSES

INTRODUCTION

You can classify segments into as many as 64 different "segment classes" or sets of segments. (The segment classes are numbered from 1 to 64.)

Once you have put segments in those classes, you can manipulate whole classes of segments with only a few commands, rather than issuing separate commands to manipulate every one of the affected segments.

For example, you can make all segments in class 13 invisible. Or, you can highlight all segments which belong to class 1 and class 2, but do not belong to class 3.

In a electronic drafting application, segment class 1 might include all segments which represent resistors. Segment class 2 might include all the segments representing capacitors. Segment class 9 might include all segments representing "new" components (components added during the last engineering change).

You might then issue a command to set the "current segment matching class" to include all segments representing resistors added during the last engineering change. This would be a <set-current-matching-class> command specifying that "segment −3" means "all segments which belong to segment class 1 and also belong to segment class 9."

With the "current segment matching class" set that way, you could then issue a command to highlight all segments in the current matching class (that is, to highlight all resistors added during the last engineering change). This would be a <set-segment-highlighting: −3, 1> command.

PROCEDURE FOR USING SEGMENT CLASSES

To use the terminal's "segment class" features, do the following:

1. Use the <set-segment-class> command to assign segments to segment classes. Each segment class includes exactly those segments which have been assigned to it with the <set-segment-class> command. There may be as many as 64 segment classes, numbered from 1 to 64.

2. Use the <set-current-matching-class> command to define the segment matching class. The current matching class is defined in terms of other (numbered) segment classes. (You cannot place a segment directly in the matching class; however, you can place it in a numbered class, and then define the segment matching class so as to include all segments in that numbered segment class.)

3. In commands to set segment attributes, you can refer to all segments in the current matching class by using the special segment number, −3.

<SET-SEGMENT-CLASS> COMMAND

To place a segment in a segment class (or remove it from a segment class), you use the <set-segment-class> command. This command has the following syntax:

<set-segment-class>
= (ESC)(S)(A) <int><int-array><int-array>

The first parameter, an <int> parameter, is the segment number.

The second parameter, an <int-array>, is the "removal array." It lists the segment classes from which the specified segment is to be removed.

The third parameter, another <int-array>, is the "addition array." It lists the segment classes to which the specified segment is to be added.

The operation of the <set-segment-class> command is as follows. The segment specified in the first parameter is removed from (made a non-member of) all the segment classes listed in the removal array. Once that is done, the segment is added to (made a member of) all the classes listed in the addition array.

For instance, to make segment one a member of exactly classes 1, 5, and 7, you could issue the following command:

<set-segment-class: 1, (−1), (1,5,7)>

= (ESC)(S)(A)
  <int: 1>
  <int-array: (−1)>
  <int-array: (1,5,7)>

= (ESC)(S)(A)(1)(1)(!)(3)(1)(5)(7)

Here, segment one is first removed from "segment class −1," that is, from all segment classes. Then it is made a member of segment classes 1, 5, and 7.

If you want to remove segment one from segment class 5, but make it a member of class 6, you can issue the following command:

<set-segment-class: 1, (5), (6)>

= (ESC)(S)(A)<int: 1>
  <int-array: (5)><int-array: (6)>

= (ESC)(S)(A) (1) (1)(5) (1)(6)

Here, the removal array has one class number, namely 5. The addition array also has one class number, namely 6. Thus segment one is removed from class 5 and added to class 6. It now belongs to exactly classes 1, 6, and 7.

Using Special Segment Numbers

In the <set-segment-class> command, you can use the special segment numbers −1, −2, and −3. For instance, you can include *all* segments in segment class 9 as in the following command:

<set-segment-class: −1, (empty array), (9)>

= (ESC)(S)(A)<int: −1>
  <int-array: empty>
  <int-array: (9)>

= (ESC)(S)(A) (!) (0) (1)(9)

Here, the first parameter specifies segment −1, meaning "all segments." The removal array is empty, so the command does not remove "all segments" from any class. The addition array holds a single class number, 9; thus all segments are added to class 9.

Again, by specifying segment −2, you can cause any segments which may be defined later to be members of certain segment classes. (On power-up, "segment −2" — the default for new segments — does not belong to any segment class.)

For example:

```
<set-segment-class: −2, (−1), (1,2,3)>
<begin-segment: 1>
   <enter-vector-mode>
      <xy>
      <xy>
      .
      .
      .
   <enter-vector-mode>
      <xy>
      <xy>
      .
      .
      .
<end-segment>
<begin-segment: 2>
   .
   .
   .
<end-segment>
<set-segment-class: −2, (1,2), (4,5,6)>
<begin-segment: 3>
   .
   .
   .
<end-segment>
```

Here, the first <set-segment-class> command removes "segment −2" from all segment classes, and then adds it to classes 1, 2, and 3. Since "segment −2" means "the default for new segments," when segments 1 and 2 are created, they are automatically included in segment classes 1, 2 and 3.

The next <set-segment-class> command removes "segment −2" from classes 1 and 2 and then adds it to classes 4, 5, and 6. Thus, when segment 3 is defined, it automatically becomes a member of segment classes 3, 4, 5, and 6.

By specifying segment −3 in the <set-segment-class> command, you can cause all segments in the current matching class to be added to or deleted from specified numbered segment classes. For example, consider the following command:

<set-segment-class: −3, (4,5,6), (1,2,3)>

= (ESC)(S)(A)<int: −3>
<int-array: (4,5,6)><int-array: (1,2,3)>

= (ESC)(S)(A) (#) (3)(4)(5)(6) (3)(1)(2)(3)

Here, the segment number is −3, meaning "all segments in the current matching class." The removal array holds the class numbers 4, 5, and 6, while the addition array holds class numbers 1, 2, and 3. This command causes all segments in the current matching class to be removed from classes 4, 5, and 6 and added to classes 1, 2, and 3.

<SET-CURRENT-MATCHING-CLASS> COMMAND

The current segment matching class is defined by the most recent <set-current-matching-class> command. That command has the following syntax:

<set-current-matching-class>
= (ESC)(S)(L) <int-array><int-array>

The first <int-array> parameter is the "inclusion array." The second <int-array> parameter is the "exclusion array."

The <set-current-matching-class> command sets the current segment matching class so as to be the class of all segments which belong to all classes in the inclusion array, but which do not belong to any classes in the exclusion array.

This can be expressed mathematically as follows. Let segment classes $A_1, A_2, \ldots, A_m$ be the classes whose class numbers are listed in the inclusion array. Let segment classes $B_1, B_2, \ldots, B_n$ be the segment classes whose numbers are listed in the exclusion array. Let segment class C be the current segment matching class. Then segment class C is the intersection of classes A_1 to A_m with the complements of classes B_1 to B_n:

$$C = A_1 \cap A_2 \cap \ldots \cap A_m \cap \bar{B}_1 \cap \bar{B}_2 \cap \ldots \cap \bar{B}_n$$

For instance, consider the following command:

<set-current-matching-class: (1,2,5), (3,7,9)>

= (ESC)(S)(L)
<int-array: (1,2,5)>
<int-array: (3,7,9)>

= (ESC)(S)(L) (3)(1)(2)(5) (3)(3)(7)(9)

This command defines the current matching class as follows: the class of all segments which belong to classes 1, 2, and 5, and which do not belong to any of the segment classes 3, 7, and 9. That is, the current matching class is the intersection of classes 1, 2, and 5 with the complements of classes 3, 7, and 9.

ADDITIONAL INFORMATION

For more information on segment classes, see the descriptions in the 4110 Series Command Reference Manual of the <set-segment-class> and <set-current-matching-class> commands.

Section 7

THE 4113 COLOR DISPLAY

INTRODUCTION

This section describes the 4113's special display features. Topics here are:

- **Color-indices and color mixtures.** The "ink bottles" in which you can dip your pen when drawing lines, and the colored "inks" with which you can fill those ink bottles.
- **Surfaces.** Dividing the 4113 display into multiple writing surfaces.
- **Selecting Color Mixtures.** Selecting the colors you will be using from the terminal's repertory of 4096 different color mixtures.
- **More about the dialog area.** Specifying the writing surface on which the dialog area appears, and the color-indices used in the dialog area.
- **Views.** Dividing the graphics display into multiple views, each with its own "viewport" on the screen.
- **View Display Clusters.** Grouping several views together into a "display cluster" to permit simultaneous "zooming" and "panning" on all views in the cluster.
- **Defining Fill Patterns.** How to define your own fill patterns for filling "panels."
- **Pixel Operations.** Commands which permit the sophisticated programmer to access directly the individual pixels in the 4113's raster memory buffer.

COLOR-INDICES AND COLOR MIXTURES

INTRODUCTION: THE COLOR RASTER-SCAN DISPLAY

The 4113 is a "raster-scan" terminal. That is, it uses a television-like display in which electron beams continually scan across the face of a cathode ray tube in a fixed horizontal-line pattern (raster). There are three electron beams controlling the brightnesses of three colored phosphors on the face of the crt (cathode ray tube).

The details of color crt design need not be described here. For the purpose of this manual, the following statements suffice:

- The three electron beams together scan across the screen.
- As the beams scan across each line, their intensities vary, producing dots or *pixels* (picture elements) of various colors.
- The intensity of the "red" beam determines the brightness of the red phosphor for each pixel, while the intensities of the "green" and "blue" electron beams determine the brightnesses of the green and blue phosphors respectively.

Lines are drawn by adjusting the red, green, and blue brightnesses of the individual pixels. In the 4113, the raster display is 640 pixels wide by 480 pixels high.

Raster Memory Buffer

To create this raster-scan display, the 4113 circuitry continually scans a *raster memory buffer*. For every pixel on the display, there is a corresponding memory location in the raster memory buffer. The circuitry scans the raster memory buffer, pixel by pixel. From each pixel memory location, the circuitry reads a *color-index* which names the color in which that pixel is to be displayed. (A more complete description of color-indices follows shortly.)

The raster memory circuitry is organized into "bit planes," as in Figure 7-1. Each bit plane contains one binary bit for each pixel of the raster-scan display. In a standard 4113 terminal, there are three bit planes; together they can store, for each pixel of the display, color-indices in the range from 0 to 7 (binary 000 to binary 111). If the terminal is equipped with Option 21, then there are four bit planes, so that color-indices can range from 0 to 15 (binary 0000 to binary 1111).

Color-Indices and Color Mixtures

The numbers stored in the raster memory bit planes do not directly determine the brightnesses of the red, green, and blue phosphors for the different pixels. Instead, each number stored in raster memory is a *color-index*: an index into a table of red, green, and blue brightnesses. The circuitry reads color-indices from the raster memory, consults its "color map" table to learn what mixture of red, green, and blue to display for those color-indices, and then displays pixels on the screen according to the color mixtures specified in the color map table. Figure 7-2 shows the principle. The figure assumes that the terminal is equipped with four bit planes (Option 21).

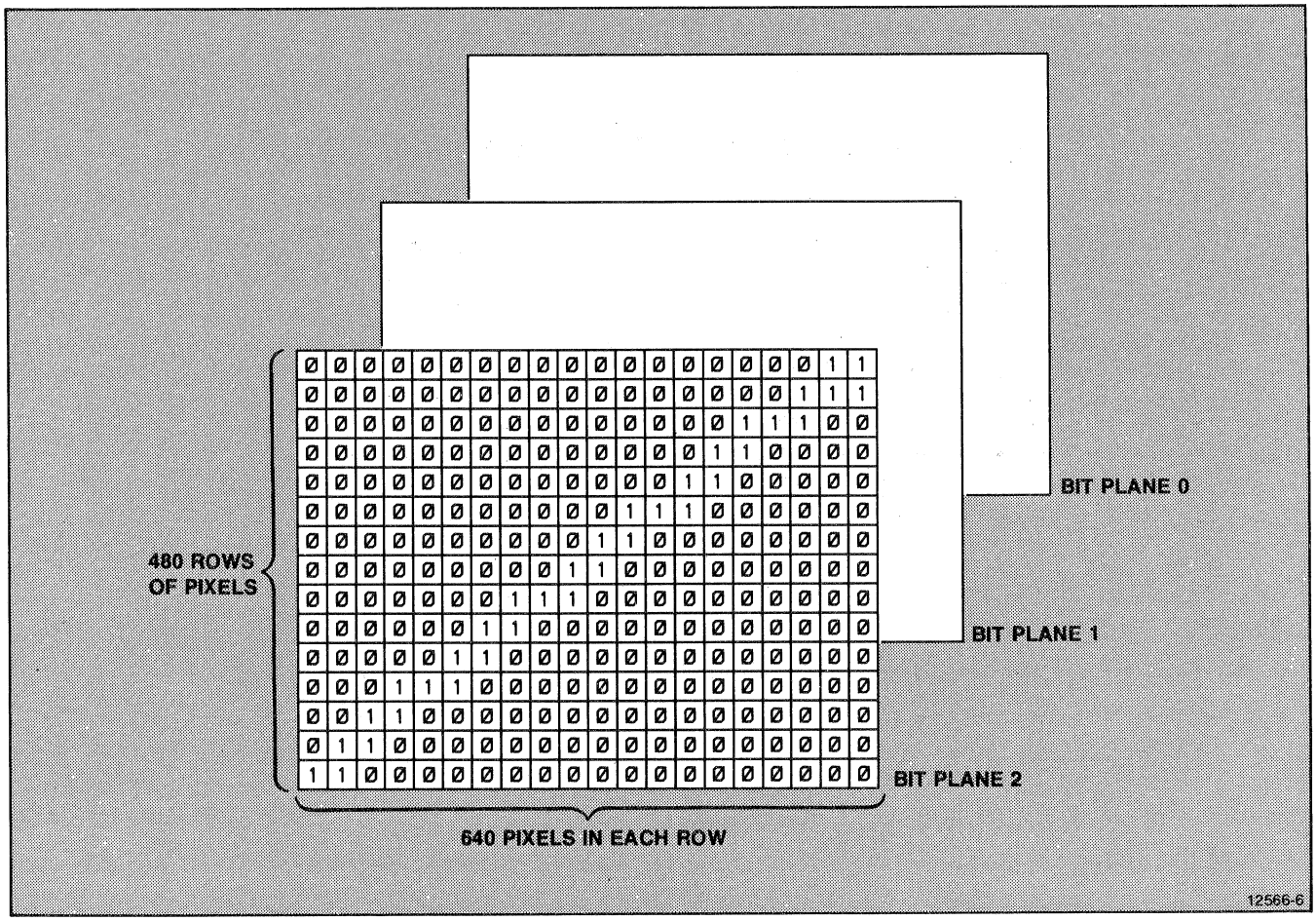


Figure 7-1. Bit Planes of Raster Memory.

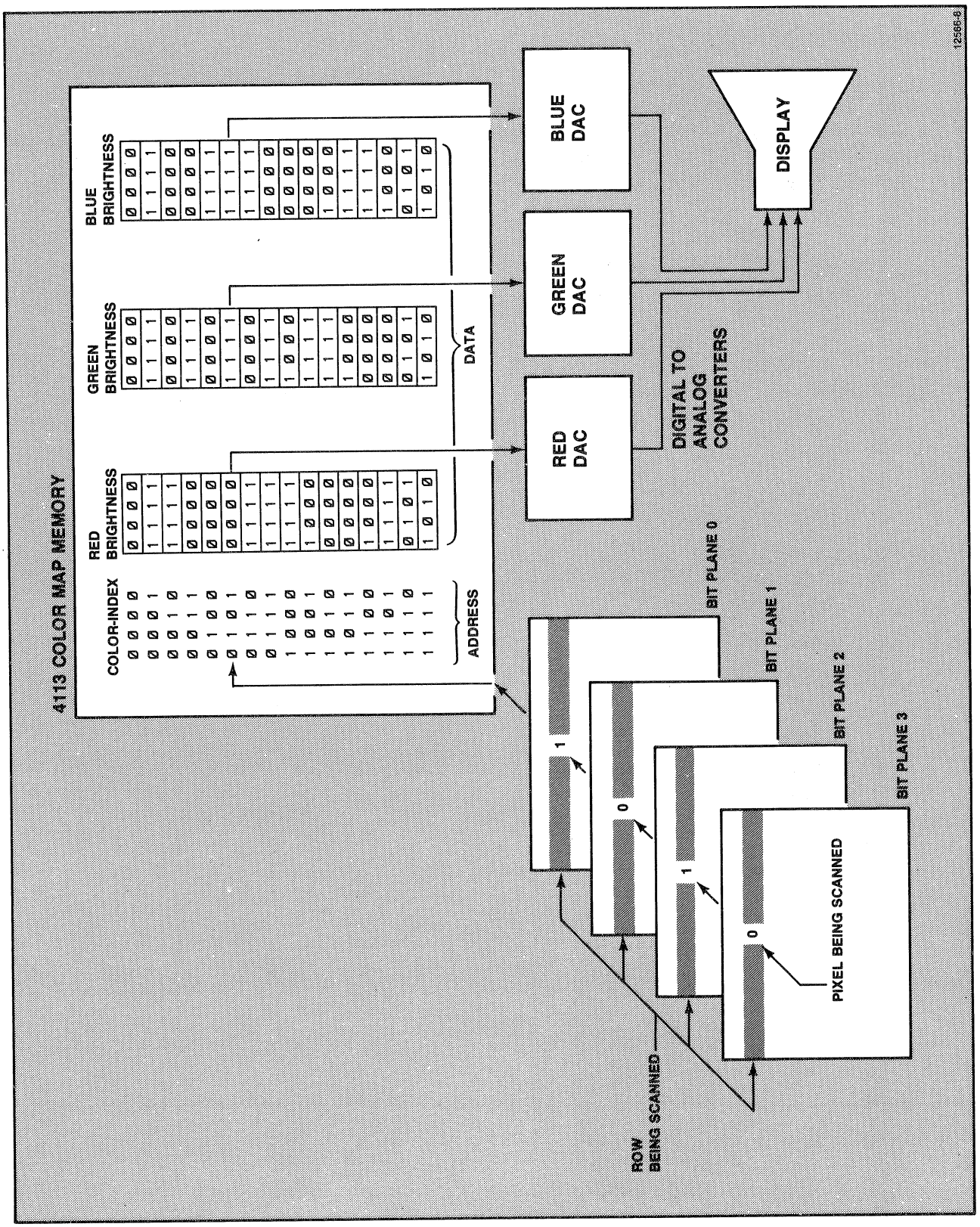


Figure 7-2. Color Display Circuitry Overall Block Diagram.

DEFAULT COLOR MIXTURES

When the terminal is turned on, its color mixtures are those in Table 7-1. (Figure 7-2 shows the color map memory with these color mixtures assigned.) You can change the color assignment with the <set-surface-color-map> command, described later in this section.

Table 7-1
DEFAULT COLOR MIXTURES

Color-Index	Color Mixture
0	Transparent (look through to the black background)
1	White
2	Red
3	Green
4	Blue
5	Cyan (Blue-Green mixture)
6	Magenta (Red-Blue mixture)
7	Yellow (Red-Green mixture)
8	Orange (Red-Yellow)
9	Yellow-Green
10	Green-Cyan
11	Cyan-Blue
12	Blue-Magenta
13	Magenta-Red
14	Dark Gray
15	Light Gray

AN ANALOGY

A Transparent Writing Surface

Think of the raster memory buffer as a transparent writing surface. Behind this surface there is a background which is all one color. (On power-up, the background is black.) Figure 7-3 shows the idea.

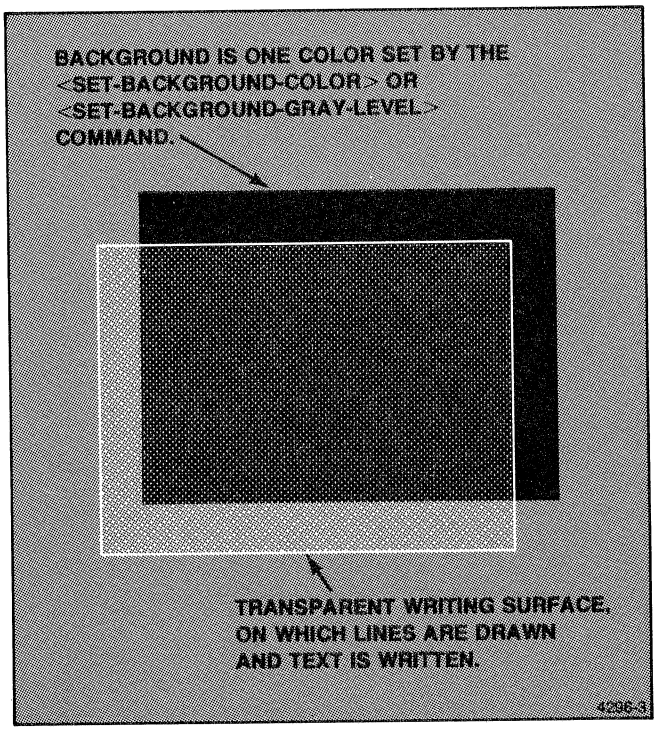


Figure 7-3. Raster Memory Space: A Transparent Writing Surface.

Inks in Ink Bottles

Think of the color-mixtures as different shades of colored ink. Since the 4113 terminal has sixteen possible brightness levels for the red, green, and blue phosphors, it has 16 to the third power, or 4096, possible color mixtures.

Think of the color-indices as ink bottles into which inks of different color mixtures may be loaded. On power-up, the standard 4113 has eight different ink bottles, numbered from 0 to 7. If the 4113 is equipped with four bit planes (Option 21), then there are sixteen ink bottles, numbered from 0 to 15. Figure 7-4 shows the analogy.

Ink bottle zero always holds "ink eradicator"; more about it later. The remaining ink bottles can each store any of the 4096 possible shades of colored ink.

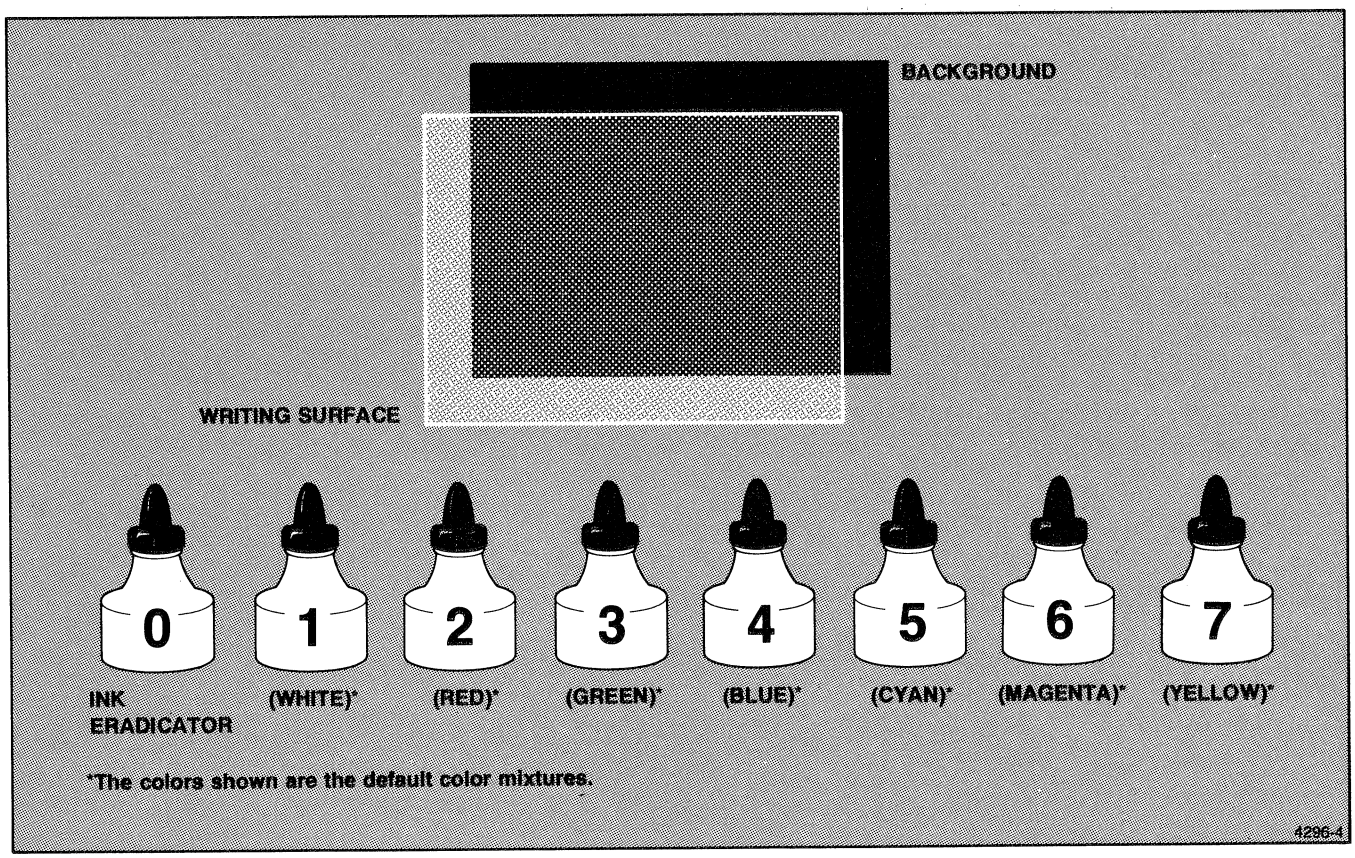


Figure 7-4. Color-Indices: Ink Bottles Holding Colored Inks.

LINE INDEX AND TEXT INDEX

To draw an image on the transparent writing surface, you first dip your pen into one of the ink bottles, and then draw lines on the writing surface. In terms of color-indices, you first select a color-index with which to write, and then draw lines (or display text) using that index. To do this, you use a <set-line-index> or <set-text-index> command. The <set-line-index> command selects the color-index (ink bottle) used for drawing lines; the <set-text-index> command selects the color-index for alphatext or graphtext.

Figure 7-5 shows an example. Here, the first command sets the line index to 7. (On power-up, color-index 7 means "yellow.") Using that color-index, the <enter-vector-mode> command and the two <xy> parameters draw a line from (1000,1000) to (3000,3000).

The next <set-line-index> command dips the pen in ink bottle number 5 (changes the current line index to color-index 5). Once that command has been executed, the following <xy> parameter causes the terminal to draw a line from the current graphic beam position to the point (3000,1000). The line is drawn using ink from bottle number 5. (On power-up, that bottle holds "cyan," or blue-green ink.)

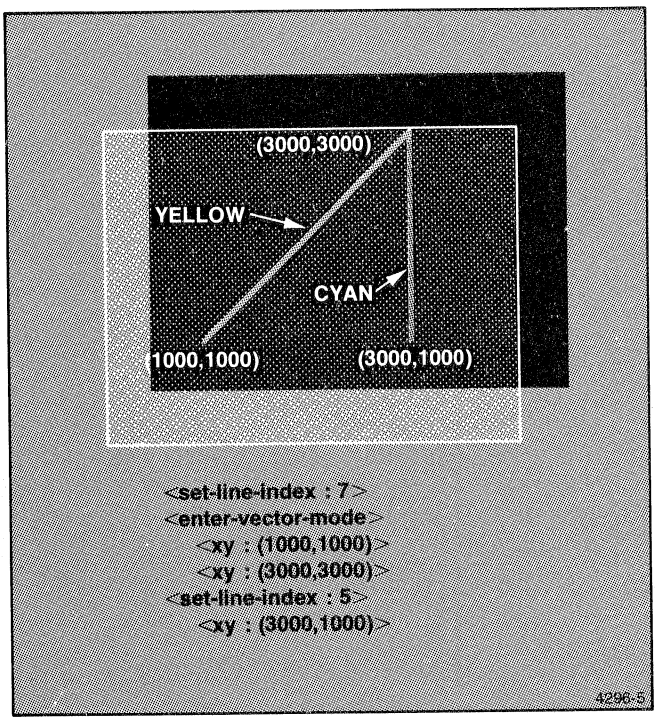


Figure 7-5. Drawing Lines in Different Color-Indices.

<SET-BACKGROUND-INDICES> COMMAND

The <set-background-indices> command specifies two additional color-indices besides the line index and text index. These are the "text background index" and the "line gap index." The syntax of the command is as follows:

<set-background-indices>
= (ESC)(M)(B)
<int: *text-background-index*>
<int: *line-gap-index*> .

Text Background Index. The first parameter determines how the background for each alphatext (or string-precision graphtext) character is displayed.

Assume that the color mixtures have their default values. Then if the current text index is one (white), and the text background index is four (blue), alphatext appears as white characters on a blue background. This applies to alphatext and graphtext outside the dialog area. (Text in the dialog area is controlled by the <set-dialog-area-indices> command, which is described later in this section.)

Line Gap Index. The second parameter in the <set-background-indices> command in the "line gap index." This index controls how the gaps in dashed lines appear.

For instance, you can draw lines which alternate between red and yellow dashes as follows:

1. Select line style seven (dashed lines) with a <set-line-style> command.
2. Set the line index to 2 (red) with a <set-line-index> command.
3. Set the line-gap-index to 7 (yellow) with a <set-background-indices> command.

Index Minus One. In the <set-background-indices> command, "index minus one" leaves pixels unchanged. This is the default on power-up for both the text background index and the line gap index.

Index Minus Two. In the <set-background-indices> command, "index minus two" specifies the wipe index for the current viewport. (The wipe index is determined by the <set-view-attributes> command. That command is described later in this section.)

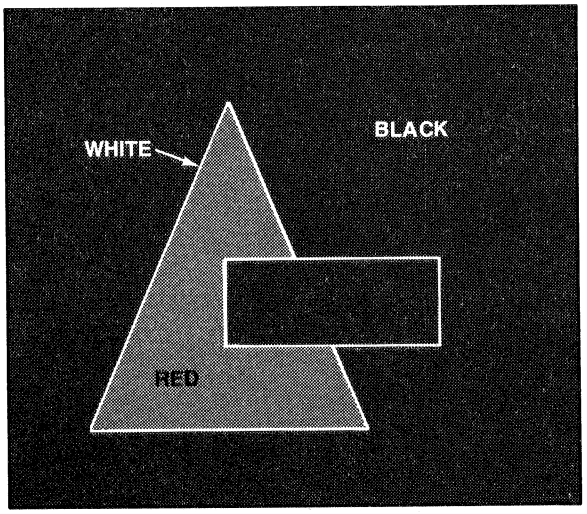
For more information, see the description of the <set-background-indices> command in the 4110 Series Command Reference Manual.

FILLING PANELS IN DIFFERENT COLORS

Drawing panels is described in Section 5. Recall from that section that the <select-fill-pattern> command lets you choose the pattern with which the interior of a panel is filled. Negative (or zero) pattern numbers select patterns whose pixels are all one color-index.

That is, in the 4113, fill patterns numbered from 0 to −15 contain pixels in color-indices 0 to 15, respectively. Thus, pattern zero always contains a transparent interior. On power-up, pattern −1 is white, pattern −2 is red, pattern −3 is green, and so on.

Figure 7-6 shows commands to draw a panel and fill it with the color red.




```
<set-panel-filling-mode : 0,0,0>
<select-fill-pattern : −2>
<enter-vector-mode>
<begin-panel-boundary>
  <xy>
  <xy>
<end-panel>
```

4296-6

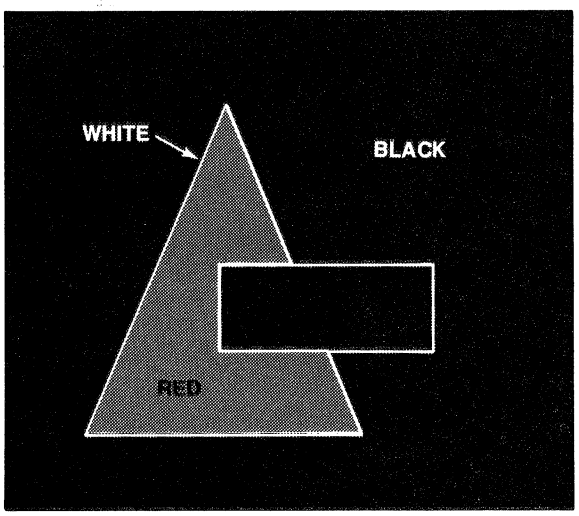
Figure 7-6. Filling a Panel With a Solid Color.

INK BOTTLE ZERO: INK ERADICATOR

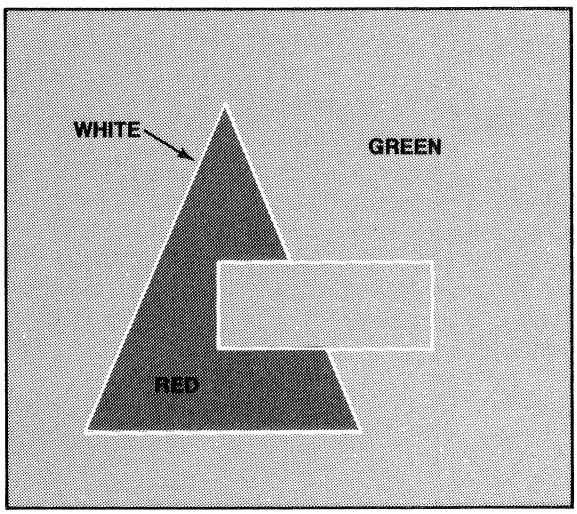
Ink bottle zero (color-index zero) always contains transparent ink, or "ink eradicator." You can erase (eradicate) a line previously drawn by drawing the line over again in color-index zero.

(More precisely, you erase a line by drawing it over again in such a way as to set its pixels to color-index zero. There is more than one way to do this. These subtleties are described later in this section.)

Figure 7-7 shows the effect of color-index zero. In the figure, a triangular panel is drawn, using fill pattern −2. This causes the panel interior to be filled with color-index 2. (The default color mixture for index 2 is red.) Then a second, rectangular panel is drawn and filled with index zero (fill pattern zero). Since index zero is "ink eradicator," the interior of this panel is transparent, showing the background behind the writing surface. In Figure 7-7A, the background is its default color (black). In Figure 7-7B, the background is another color. (Setting the background color is described later in this section.)



A. Shown Against Black Background.



B. Shown Against Colored Background.

4296-7

Figure 7-7. Erasing a Region With Color-Index Zero.

HOW COLOR-INDICES ARE WRITTEN INTO THE RASTER MEMORY BUFFER

When you draw a line, you name the end points of that line, using x- and y-coordinates which are whole numbers in the range from 0 to 4095. In other words, you draw the line in an imaginary "terminal space," which has 4096 possible x-coordinates and 4096 possible y-coordinates.

To display lines drawn in the 4096-by-4096 terminal space, the 4113 sets the color-indices of pixels in its 640-by-480 raster memory buffer. It computes which pixels in its 640-by-480 raster memory space are closest to the line, and changes the color-indices of those pixels. Circuitry in the 4113 then automatically displays the pixels in the raster memory buffer.

When drawing lines which comprise a segment, the 4113 consults a variable associated with that segment: the *segment writing mode*. (This variable is set by the <set-segment-writing-mode> command.) The segment writing mode may be either *set mode* or *XOR mode*.

Set Mode

In set mode, as the 4113 writes an image of the segment into its raster memory bit planes, each pixel in the buffer is set exactly to the appropriate color-index: the current line index for pixels which form the image of lines in terminal space, and the current text index for pixels which make up characters of alphatext or graphtext.

XOR Mode

In XOR (exclusive-OR) mode, as the 4113 writes an image of a segment into its raster memory bit planes, each pixel being written over is replaced by a pixel in a new color-index. However, this new color-index is not necessarily the current line index or text index. (It is only that index if the pixel's old color-index was zero). Instead, the new color-index is a binary number which is the bit-by-bit "exclusive-OR" of the bits in the old color-index for the pixel and the corresponding bits of the current line index or text index.

(The XOR operation on two binary bits results in 1 if the bits are different and in 0 if the bits are the same.)

For instance, if a pixel's old color-index is 6 (binary 110), and it is being overwritten by a line whose line index is 3 (binary 011), then the pixel is set to color-index 5 (binary 101). The leftmost bit is 1 because (1 XOR 0) = 1. The middle bit is 0 because (1 XOR 1) = 0. The rightmost bit is 1 because (0 XOR 1) = 1.

This XOR mode is convenient for writing images which may later need to be erased or repositioned on the screen. This is because a line can be erased by writing over it again in XOR mode. (This property is a consequence of the Boolean logic theorem that "(A XOR B) XOR B = A.").

<Set-Segment-Writing-Mode> Command

You can set a segment's writing mode with the <set-segment-writing-mode> command:

<set-segment-writing-mode> = (ESC)(S)(M)<int>

Here, the <int> is 1 if the segment is to be displayed in set mode. It is 2 if the segment is to be displayed in XOR mode. For more details, see the description of the <set-segment-writing-mode> command in the 4110 Series Command Reference Manual.

The default writing mode for all segments is set mode. When a segment is used as a graphic cursor, however, it is displayed in XOR mode.

SURFACES

INTRODUCTION

Earlier in this section (Figure 7-3), raster memory space was represented as a transparent writing surface with an opaque background behind it. For a standard 4113 (one without Option 21), there are three bit planes, and therefore three bits per pixel. With three bits per pixel, there are eight ink bottles (color-indices) in which to dip the pen when drawing lines on the transparent writing surface. If Option 21 is installed, there are four bits for each pixel, and so there are sixteen ink bottles (color-indices) available.

However, the 4113 permits another possibility. You can split the available bit planes among several writing surfaces. Figure 7-8 shows the concept, while Figure 7-9 shows hardware details. In these figures, Surface 1 (the front surface) has a one-bit plane, while Surface 2 (the rear surface), has a two-bit plane assigned to it. Thus each surface's color-indices can range only from 0 (binary 00) to 3 (binary 11).

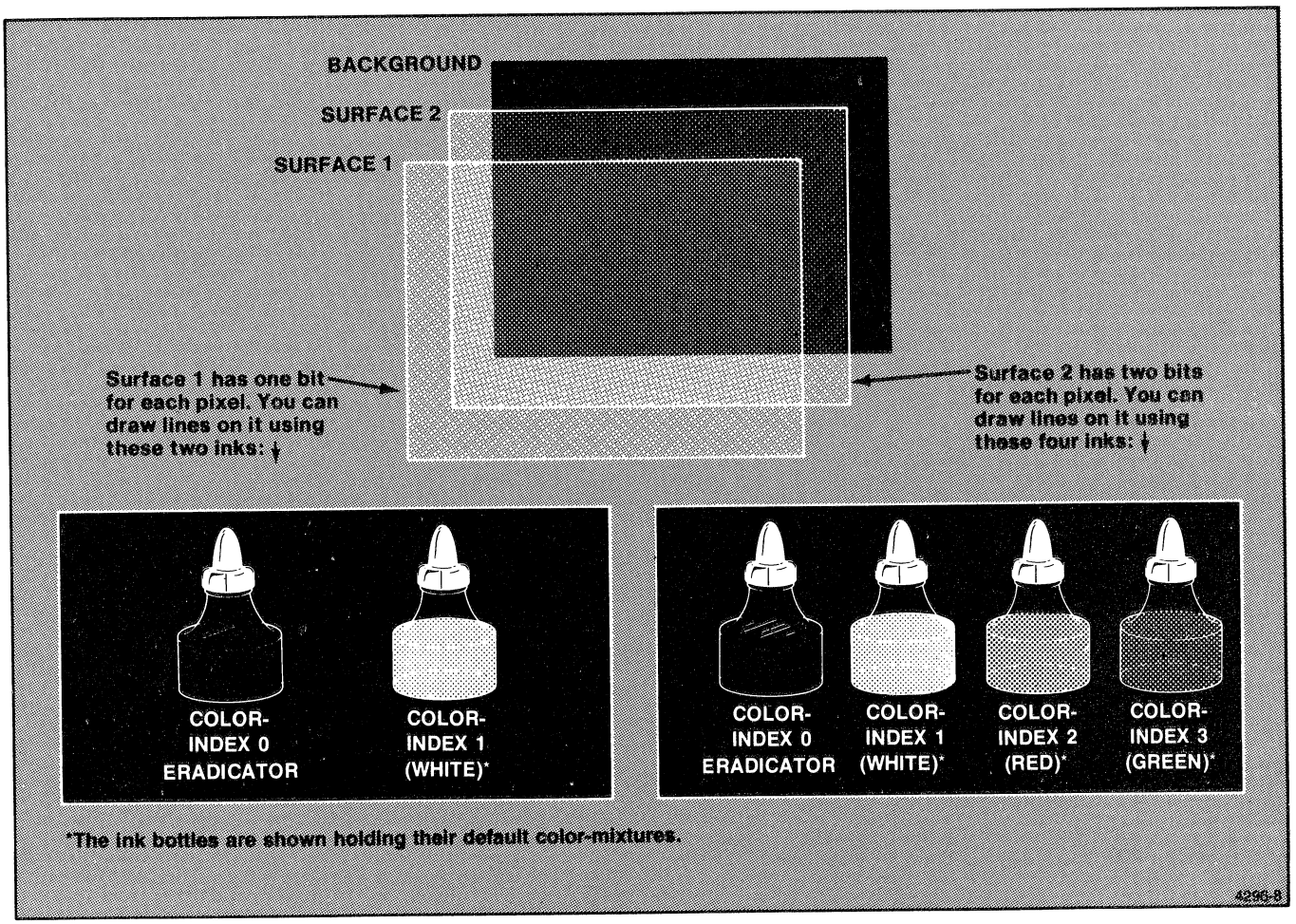


Figure 7-8. Two Surfaces and Their Ink Bottles (Color Indices).

SURFACES

When drawing lines on Surface 1, you use the ink bottles (color-indices) associated with Surface 1. When drawing on Surface 2, you use the ink bottles associated with Surface 2. The two sets of ink bottles can have different inks in them; ink bottle one for Surface 1 may hold white ink, while ink bottle one for Surface 2 may hold blue ink.

However, for both surfaces, ink bottle zero holds "ink eradicator." Pixels set to color-index zero let you to look through a surface to the surface behind it. Behind both surfaces is the "background." You can set the background color with either the <set-background-color> command or the <set-background-gray-level> command. (More about these commands follows.)

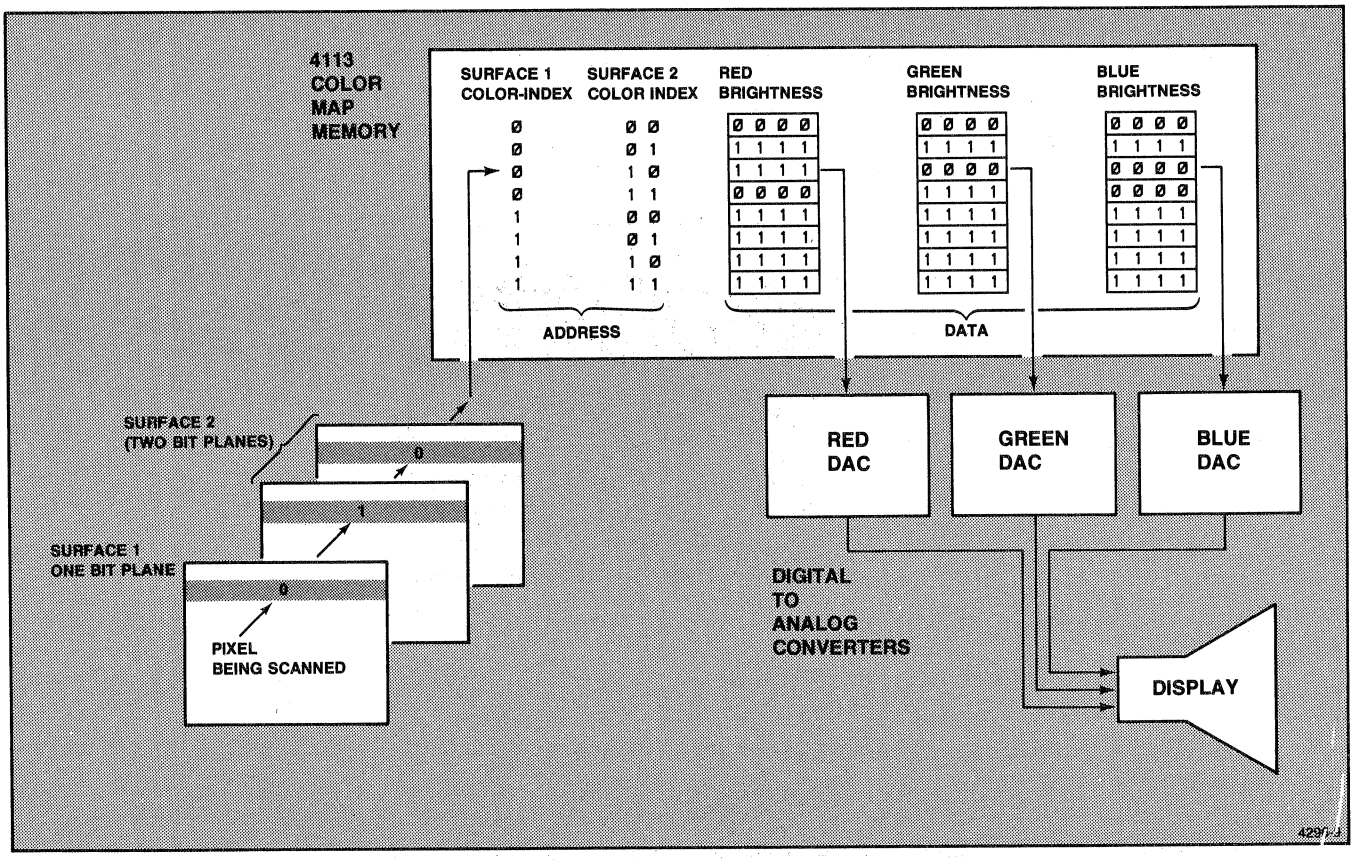


Figure 7-9. Two Surfaces: 4113 Display Details.

DEFINING SURFACES

<Set-Surface-Definitions> Command

To create more than one surface, and to allocate the available raster memory bits among the surfaces, use the <set-surface-definitions> command:

<set-surface-definitions> = (ESC)(R)(D)<int-array>

The number of integers in the <int-array> parameter tells how many writing surfaces are being defined. Each integer in the array tells how many raster memory bit planes are assigned to the corresponding surface.

<set-surface-definitions : (1,1,1,1)>

= (ESC)(R)(D)
<int-array : (1,1,1,1)>

= (ESC)(R)(D)
(4)(1)(1)(1)(1) .

Defines four surfaces, each with one bit plane. On each of these surfaces, the allowable color-indices are 0 and 1.

<set-surface-definitions : (1,2)>

= (ESC)(R)(D)<int-array : (1,2)>

= (ESC)(R)(D)(2)(1)(2) .

Defines two surfaces. Surface 1 has one bit plane; its color-indices are 0 and 1. Surface 2 has two bit planes; its color-indices are 0, 1, 2, and 3. (If the terminal has four bit planes—Option 21—the extra bit plane is unused.)

<set-surface-definitions : (2,1)>

= (ESC)(R)(D)<int-array : (2,1)>

= (ESC)(R)(D)(2)(2)(1) .

Defines two surfaces. Surface 1 has two bit planes, and permits color-indices 0, 1, 2, and 3. Surface 3 has one bit plane; its only color-indices are 0 and 1.

<set-surface-definitions : (4)>

= (ESC)(R)(D)<int-array : (4)>

= (ESC)(R)(D)(1)(4) .

Defines one surface with four bit planes. Color indices on Surface 1 can range from 0 to 15.

Besides allocating bit planes among the surfaces, the <set-surface-definitions> command also sets default color mixtures for each surface's color-indices. In other words, it fills each surface's ink bottles with inks of certain colors. These are the same color mixtures that were listed in Table 7-1 for the power-up condition.

For instance, on a one-bit-plane surface, the <set-surface-definitions> command sets the color mixtures as follows:

Color-Index	Color Mixture
0	Transparent (ink eradicator)
1	White

Again, on a two-bit-plane surface, the color mixtures are set as follows:

Color-Index	Color Mixture
0	Transparent (ink eradicator)
1	White
2	Red
3	Green

On a three-bit-plane surface, the mixtures are set to these values:

Color-Index	Color Mixture
0	Transparent (ink eradicator)
1	White
2	Red
3	Green
4	Blue
5	Cyan
6	Magenta
7	Yellow

On a four-bit-plane surface (available only with Option 21), all the mixtures in Table 7-1 are available.

If you want to use different color mixtures than these, you can select your own color mixtures with a <set-surface-color-map> or <set-surface-gray-levels> command. These commands are described later in this section.

<Set-Surface-Priorities> Command

The <set-surface-definitions> command defines the surfaces from front to back, with Surface 1 always in front. If you like, you can rearrange the order of the surfaces with a <set-surface-priorities> command. For instance, you can place Surface 2 in front of Surface 1 with the following command:

<set-surface-priorities: (2,1,1,2)>

= (ESC)(R)(N)<int-array: (2,1,1,2)>

= (ESC)(R)(N)(4)(2)(1)(1)(2)> .

The numbers in the command's <int-array> parameters are grouped in pairs. The first pair of numbers specify that Surface 2 is to have priority 1 (to be in front of other surfaces). The second pair of numbers specify that Surface 1 is to have priority 2 (to be the second surface in the front-to-back ordering).

For more information, see the description in the Command Reference Manual of the <set-surface-priorities> command.

Surface Visibility

The <set-surface-visibility> command lets you control whether an entire surface is visible or invisible:

<set-surface-visibility>

= (ESC)(R)(I)
<int: *surface-number*>
<int: *visibility-mode*> .

In this command, the first <int> parameter is the surface number, while the second parameter controls whether the entire surface is to be visible or invisible. If this latter parameter is zero, the surface is invisible; objects drawn on it are not displayed. If the parameter is one, the surface is visible. If the parameter is two, the surface "blinks"—alternates between being visible and being invisible.

Undefined Surfaces

Returning now to the <set-surface-definitions> command, recall that the size of that command's <int-array> determines which surfaces are defined. For instance, if the <int-array> holds three <int>s, then there are three surfaces, numbered from 1 to 3.

If you issue a <set-surface-definitions> command which defines two surfaces, then those are the only two surfaces in existence. Surface 3 is undefined. If you later try to refer to Surface 3, the terminal will detect an error.

Surfaces with Zero Bit Planes

An undefined surface, however, is not quite the same as a surface with zero bit planes. It is possible, in a <set-surface-definitions> command, to assign zero bit planes to one or more surfaces. Consider, for example, the following command:

<set-surface-definitions: (1,0,2)>

= (ESC)(R)(D)<int-array: (1,0,2)>

= (ESC)(R)(D)(3)(1)(0)(2) .

Since there are three numbers in the <int-array>, three surfaces are defined. Surface 1 has one bit plane. Surface 2 has *zero* bit planes. Surface 3 has two bit planes.

Since Surface 2 has zero bit planes, it cannot be displayed on the screen. However, it is still possible to assign a viewport to Surface 2, and to draw images in that imaginary viewport. (Views and viewports are described later in this section.) Having drawn an image on Surface 2, you can then issue another <set-surface-definitions> command:

<set-surface-definitions: (1,2,0)>

= (ESC)(R)(D)<int-array: (1,2,0)>

= (ESC)(R)(D)(3)(1)(2)(0) .

Now that two bit planes have been assigned to Surface 2, the terminal can display the image just drawn on that surface. However, Surface 3 now becomes invisible, because it no longer has any bit planes.

To use zero-bit-plane surfaces properly requires that viewports be assigned to those surfaces. Therefore, an example of how to do this is deferred until later in this section, after views and viewports have been described.

SELECTING COLOR MIXTURES

As described earlier in this section, there are certain default color mixtures which are available on power-up, or when a <set-surface-definitions> command is issued. If you want to use different color mixtures than these, you can select them with the following commands:

<Set-Surface-Gray-Levels>	Assigns "gray" color mixtures (gray inks) to the color-indices (ink bottles) for a particular surface.
<Set-Background-Gray-Level>	Assigns a gray color mixture to the background that is behind all the surfaces.
<Set-Surface-Color-Map>	Assigns color mixtures (colored inks) to the color-indices (ink bottles) for a particular surface.
<Set-Background-Color>	Assigns a color mixture to the background that is behind all the surfaces.

Each of these commands is described in turn.

<SET-SURFACE-GRAY-LEVELS> COMMAND

Of the 4096 possible color mixtures in the 4113 terminal, just 16 color mixtures are shades of gray. These are represented by "lightness levels" from 0% (black) to 100% (white). Figure 7-10 shows the lightness levels for each of these sixteen shades of gray.

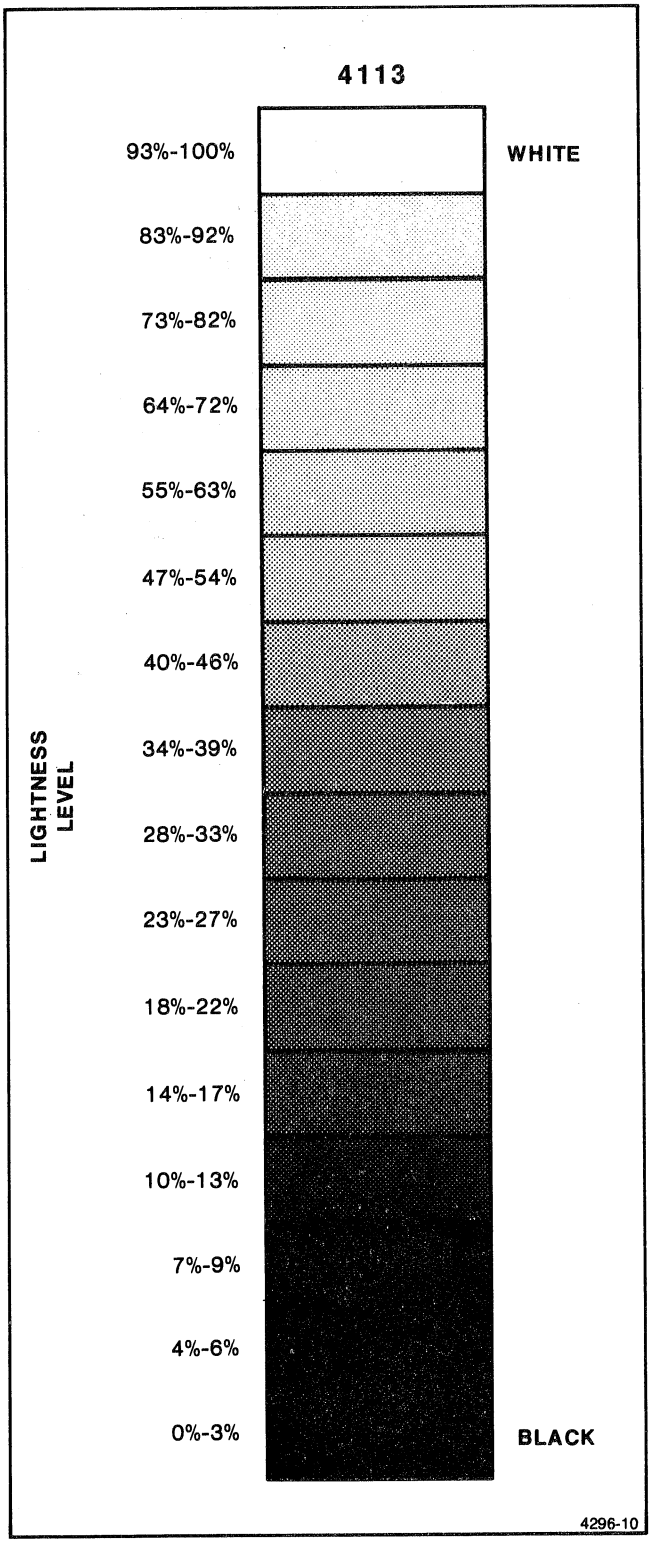


Figure 7-10. Shades of Gray Available in the 4113.

To assign shades of gray to different color-indices, you issue the <set-surface-gray-levels> command. This command has the following syntax:

<set-surface-gray-levels>

= (ESC)(R)(G)
<int : *surface-number*>
<int-array : *color-indices-and-gray-levels*> .

The first parameter is an <int> naming the surface for which you are specifying the gray-levels. (Remember, each surface has its own set of ink-bottles.)

The second parameter is an array holding an even number of <int>s, arranged in pairs. The first <int> in each pair is a color-index; the following <int> is the corresponding gray-level (in the range from 0 to 100).

For example, suppose that on Surface 1 you want to assign color-index 1 to a gray-level of 90% lightness, to assign color-index 2 to gray-level 75%, and to assign color-index 3 to gray-level 100%. Then you would issue the following command:

<set-surface-gray-levels : 1, (1,50,2,75,3,100)>

= (ESC)(R)(G)<int : 1>
<int-array : (1,50,2,75,3,100)>

= (ESC)(R)(G)
<int:1>
<int:6><int:1><int:50><int:2><int:75>
<int:3><int:100>

= (ESC)(R)(G)
(1)
(6) (1) (C)(2) (2) (D)(;) (3) (F)(4)

= (ESC)(R)(G)(1)(6)(1)(C)(2)(2)(D)(;)(3)(F)(4) .

<SET-BACKGROUND-GRAY-LEVEL> COMMAND

To set the background color to a shade of gray, you can use the <set-background-gray-level> command:

<set-background-gray-level>

= (ESC)(R)(B)
<int: *gray-level*> .

Here, the gray-level parameter is a lightness level, between 0 and 100. On power-up, the background is black (0% lightness).

THE HLS COLOR COORDINATE SYSTEM

The <set-surface-color-map> and <set-background-color> commands specify color mixtures with *color coordinates*. The 4113 terminal can use any of three color coordinate systems: RGB, CMY, and HLS. These systems are described in Appendix D.

The HLS (hue, lightness, saturation) coordinate system is the terminal's default system, in effect when the terminal is turned on. To prepare you for the <set-surface-color-map> and <set-background-color> commands, the HLS system is described here. For more information on this and other color coordinate systems, see Appendix D.

In the HLS coordinate system, the world of possible color mixtures is represented as a double-ended cone (Figure 7-11). (The cone is shown in color in Appendix D.)

Hue. The H, or hue, coordinate runs around the cone. It is expressed as an angle from 0 to 360 degrees:

H Coordinate	Hue
0	Blue
60	Magenta
120	Red
150	Orange
180	Yellow
210	Yellow-Green
240	Green
300	Cyan

Lightness. The L, or lightness, coordinate runs vertically up the cone. Black is at the bottom, 0% lightness. White is at the top, 100% lightness.

Saturation. The S, or saturation, coordinate runs radially outward from the axis of the cone. Points on the axis of the cone (0% saturation) represent shades of gray. Points on the surface of the cone (100% saturation) represent the most fully saturated color mixtures obtainable at their lightness levels. (The S coordinate expresses saturation as a percentage of the maximum possible saturation at a given lightness level.) The most fully saturated color mixtures (the reddest reds, the greenest greens, etc.) lie on the surface of the cone (S= 100) at the 50% lightness level, where the HLS double-ended cone is widest.

For more information, see Appendix D.

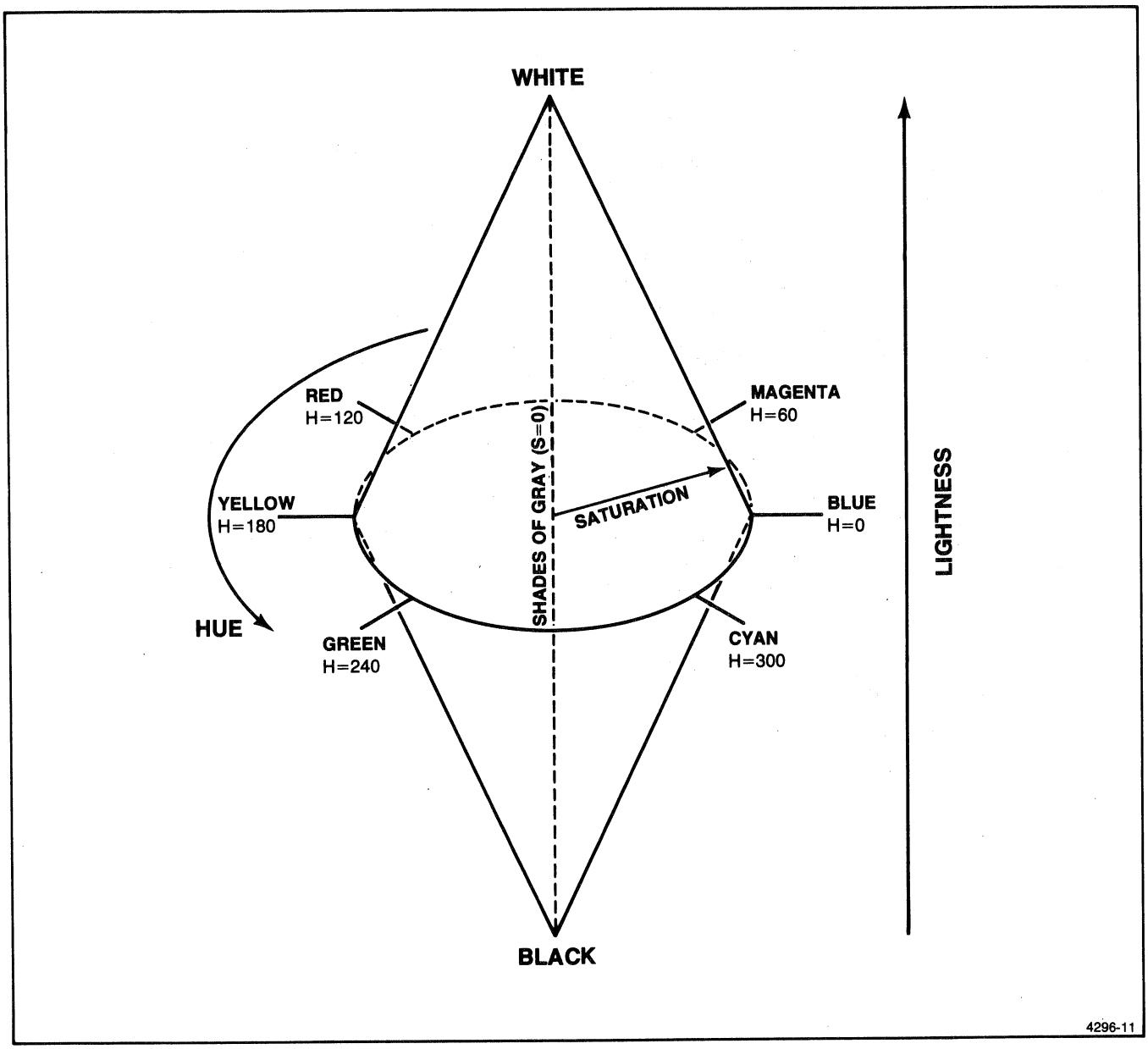


Figure 7-11. HLS Color Coordinate System.

<SET-SURFACE-COLOR-MAP> COMMAND

The <set-surface-color-map> command loads the ink bottles for a particular surface with colored inks. That is, the command assigns color mixtures to each of a surface's non-zero color-indices. The command has the following syntax:

<set-surface-color-map>

= (ESC)(T)(G)
<int: *surface-number*>
<int-array: *indices-and-color-coordinates*> .

In this command, the <int> parameter names the surface for which color-mixtures are being specified.

The <int-array> parameter contains groups of four numbers. The first number in each group is a color-index, while the other three numbers are color coordinates. On power-up, these are HLS (hue, lightness, saturation) coordinates. However, other color coordinate systems can be selected with the <set-color-mode> command, described later in this section.

Example

Suppose the HLS color coordinate system is being used, and that the terminal has four bit planes (Option 21). With the <set-surface-definitions> command, you can define two surfaces, of two bit planes each. That command also selects default color mixtures for these surfaces. You can then change the color mixtures for one of the surfaces, so that you can tell by an object's color on which surface it is located. To do this, you might issue the following commands:

<set-surface-definitions: (2,2)>

= (ESC)(R)(D)<int-array: (2,2)>

= (ESC)(R)(D)(2)(2)(2) .

<set-surface-color-map>

= (ESC)(T)(G)
<int: *surface-number*>
<int-array: *indices-and-color-mixtures*>

= (ESC)(T)(G)
<int: 2>
<int-array: (1, H_1, L_1, S_1,
2, H_2, L_2, S_2,
3, H_3, L_3, S_3)>

= (ESC)(T)(G)
<int:2>
<int-array: (1,300,50,100,
2,60,50,100,
3,180,50,100)>

= (ESC)(T)(G)(2)(<)(1)(R)(<)(C)(2)(F)(4)
(2)(C)(<)(C)(2)(F)(4)(3)(K)(4)(C)(2)(F)(4) .

In this example, the <set-surface-definitions> command defines two surfaces, of two bit planes each. On both surfaces, it sets the color mixtures as follows: index 1, white; index 2, red; index 3, green.

The <set-surface-color-map> changes the color mixtures for Surface 2. It sets the Surface 2 colors as follows:

- Index 1 is set to "cyan." This color mixture has these coordinates: H = 300, L = 50, S = 100.
- Index 2 is set to "magenta." H,L,S = 60, 50, 100.
- Index 3 is set to "yellow." H,L,S = 180, 50, 100.

SETUP Mode: CMAP Command

The SETUP mode name for the <set-color-map> command is CMAP. Thus, in SETUP mode, the operator could set Surface 2's color-indices to cyan, magenta, and yellow as follows:

```
CMAP  2  1,300,50,100,2,60,50,100,3,180,50,100
```

This could also be typed in three lines, as follows:

```
CMAP  2  1,300,50,100
CMAP  2  2,60,50,100
CMAP  2  3,180,50,100
```

<SET-BACKGROUND-COLOR> COMMAND

The <set-background-color> command sets the color mixture for the "background," which is behind all surfaces. Its syntax is as follows:

<set-background-color>

= (ESC)(T)(B)
<int: *first-color-coordinate*>
<int: *second-color-coordinate*>
<int: *third-color-coordinate*> .

The color coordinates are those for the current color coordinate system, as selected by the most recent <set-color-mode> command. On power-up, this is the HLS coordinate system, so on power-up the <int> parameters represent hue, lightness, and saturation, respectively.

For instance, to set the background to magenta (H= 60, L= 50, S= 100), you can issue the following command:

<set-background-color: 60, 50, 100>

= (ESC)(T)(B)
<int: 60>
<int: 50>
<int: 100>

= (ESC)(T)(B)(C)(<)(C)(2)(F)(4) .

<SET-COLOR-MODE> COMMAND

The <set-color-mode> command lets you control three aspects of the terminal's color display system. These are: the color coordinate system, the color overlay mode, and whether the display is in color or in black and white. The syntax of the command is as follows:

<set-color-mode> = (ESC)(T)(M)
<int: *color-specifying-mode*>
<int: *color-overlay-mode*>
<int: *gray-mode*> .

Color Specifying Mode. The first <int> parameter is in the range from 0 to 3. It specifies the color coordinate system to be used, as follows:

0 = no change
1 = RGB (red, green, blue)
2 = CMY (cyan, magenta, yellow)
3 = HLS (hue, lightness, saturation)

On power-up, the HLS coordinate system is in effect. For more information on these color coordinate systems, see Appendix D.

Color Overlay Mode. The second <int> parameter is also in the range from 0 to 3. It specifies the "overlay mode," as follows:

0 = no change
1 = OPAQUE
2 = SUBTRACTIVE
3 = ADDITIVE

In OPAQUE mode (mode 1), pictures drawn on a surface are deemed to be opaque; they obscure pictures drawn on surfaces behind them. When the terminal is turned on, it is in OPAQUE mode.

In SUBTRACTIVE mode (mode 2), pictures are drawn using transparent inks. The terminal behaves like a light table, in which transparent overlays are placed on top of a diffuse light source. (That light source would be the background behind all the surfaces.)

NOTE

For SUBTRACTIVE mode to work properly, the background color should be white or some other light color.

In ADDITIVE mode (mode 3), the images drawn on different surfaces act as if their colored inks were composed of many small light sources. Where colors on one surface overlap colors on another surface, the light coming from the first surface combines with the light coming from the second surface. For instance, light from a red object on one surface can combine with light from an overlapping green object on another surface to provide a yellow color where the objects overlap.

Gray Mode. The <set-color-mode> command's third parameter determines whether colored objects are displayed in color, or in black and white. If this parameter is zero (or is omitted), the gray mode is left unchanged. COL mode (mode 1) causes the 4113 to operate normally as a color graphics terminal. This is the mode in effect when the terminal is turned on. BW mode (mode 2) causes colors to appear as shades of gray, as if viewed on a black-and-white television set.

SETUP Mode: CMODE Command

The SETUP mode name for the <set-color-mode> command is CMODE. Thus, in SETUP mode, the operator can change the terminal's color modes as follows:

CMODE RGB 0 0 *or* CMODE RGB	Change the color coordinate system to the RGB system, without changing the other color mode settings.
CMODE 0 SUBTRACTIVE 0 *or* CMODE 0 SUBTRACTIVE	Change the overlay mode to SUBTRACTIVE mode, without affecting the other color mode settings.
CMODE 0 0 BW	Change the gray mode to BW, without affecting the other color mode settings. Colors will be displayed in shades of gray, as if on a black-and-white television set.
CMODE HLS OPAQUE COL	Sets the three color mode settings back to their original values: HLS color coordinates, OPAQUE overlay mode, and COL display mode.

For more information on the SETUP mode CMODE command, see the 4113 Operator's Manual.

MORE ABOUT THE DIALOG AREA

The concepts of color-indices and surfaces apply to the dialog area as well as to graphics. Recall that the dialog area scroll is displayed in a portion of the screen called the dialog viewport. You can assign the dialog viewport to a particular surface with the <set-dialog-area-surface> command. Also, you can specify several color-indices used in the dialog area, with the <set-dialog-area-index> command.

<SET-DIALOG-AREA-SURFACE> COMMAND

The <set-dialog-area-surface> command has this syntax:

<set-dialog-area-surface> = (ESC)(L)(S)<int> .

Here, the <int> parameter specifies the writing surface on which the dialog area viewport is to be displayed.

One way to use this command is to place the dialog area on one surface, and to display graphics on another surface. You could, for instance, issue the following commands:

```
<set-surface-definitions : (2,1)>
<set-surface-priorities : (1,2,2,1)>
<set-dialog-area-surface : 2>
<set-dialog-area-lines : 30>
<set-dialog-area-chars : 80>
<set-dialog-area-position : (0,0)>
<enable-dialog-area : 1>
<set-dialog-area-visibility : 1>
```

Here, the <set-surface-definitions> command creates two surfaces. On power-up, all graphics goes to Surface 1. So, Surface 1 is assigned two bit planes, letting it display lines in four different color-indices. Surface 2 will be used for the dialog area; it is assigned one bit plane.

The <set-surface-priorities> command rearranges the surfaces, placing Surface 1 behind Surface 2. Thus the dialog area, when displayed on Surface 2, will not be obscured by graphics drawn on Surface 1.

The next four commands define the dialog area's viewport. The <set-dialog-area-surface> command places the dialog viewport on Surface 2. The <set-dialog-area-lines> and <set-dialog-area-chars> commands define the extent of the dialog area on that surface. The <set-dialog-area-position> command places the lower left corner of the dialog area at the lower left corner of the writing surface.

The final two commands enable the dialog area and make it visible. Alphatext from the host is now directed to the dialog area, on Surface 2 (the front surface in this example). Graphics, however, goes to Surface 1 rear surface). If the operator wants to look at the graphics unobscured by text from the dialog area, he or she presses the DIALOG key to make the dialog area invisible. To view text in the dialog area again, the operator presses DIALOG again to make the dialog area visible.

Figure 7-12 shows the effect of these commands.

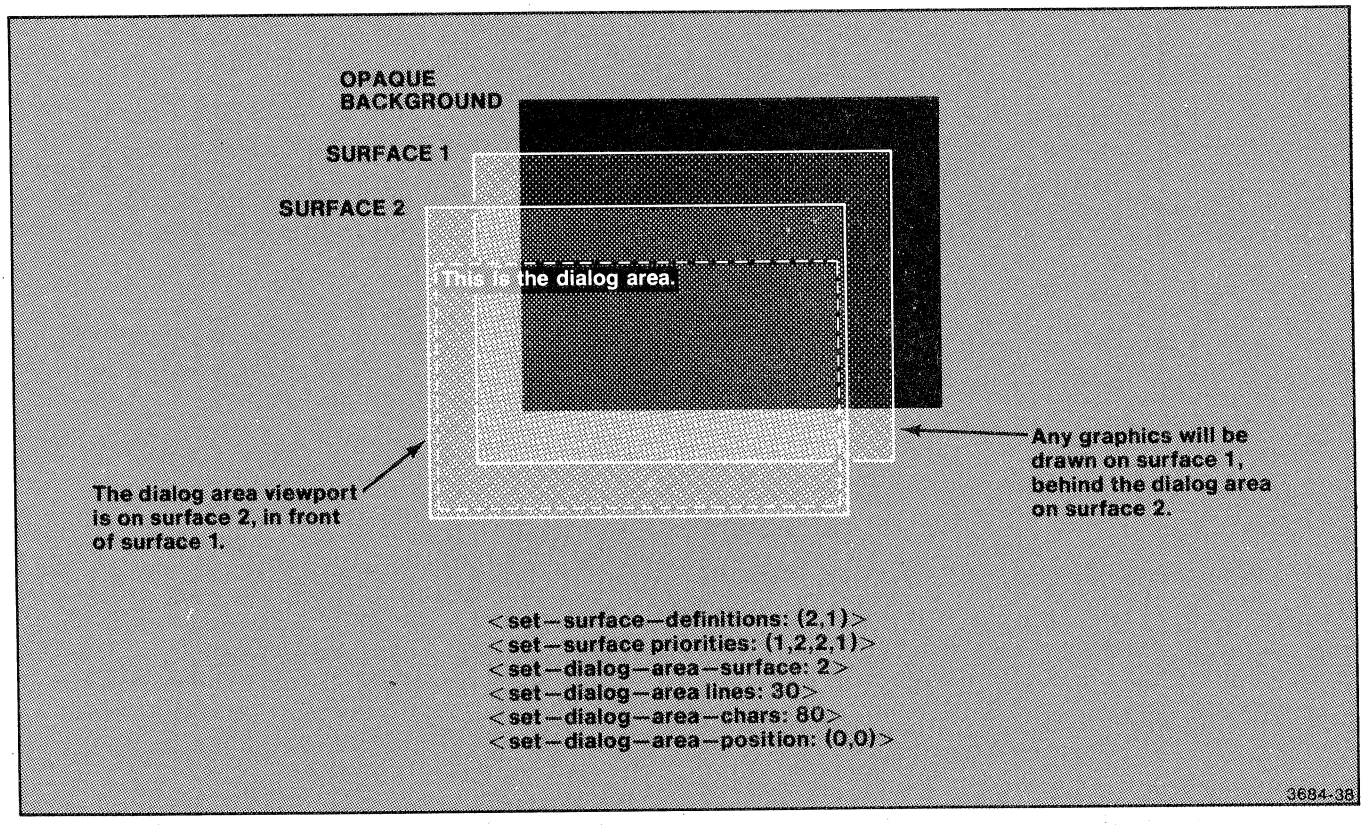


Figure 7-12. Putting the Dialog Area on Its Own Separate Surface.

<SET-DIALOG-AREA-INDEX> COMMAND

You can specify three color-indices for the dialog area with the <set-dialog-area-index> command:

<set-dialog-area-index> = (ESC)(L)(I)
<int : *character-index*>
<int : *background-index*>
<int : *wipe-index*> .

The first integer parameter specifies the color-index in which characters are displayed. The second parameter specifies the index in which the background of each character cell is displayed. The third parameter specifies the index to which all pixels in the dialog viewport are set when the dialog area is erased by the <clear-dialog-scroll> command or the CLEAR key.

If no <set-dialog-area-index> command has been issued, then default values for these indices are assumed. The default character index is one, while the default background and wipe indices are both zero.

The following commands create a dialog area for which the character index, background index, and wipe index are all different:

<set-surface-definitions : (2,1)>
<set-dialog-area-surface : 1>
<set-dialog-area-lines : 30>
<set-dialog-area-chars : 80>
<set-dialog-area-position : (0,0)>
<set-dialog-area-index : 1,2,0>
<enable-dialog-area : 1>
<set-dialog-area-visibility : 1>
<enter-alpha-mode>
(D)(i)(a)(l)(o)(g)(SP)(A)(r)(e)(a)(CR)(LF)

Here, the character index is set to 1, whose default color mixture is white. The background index is 2, whose default color is red. The dialog area wipe index is zero (transparent). Thus you can look through the unused parts of the dialog area (on Surface 1) to see whatever graphics may be displayed on Surface 2 behind.

Figure 7-13 shows the result of these commands.

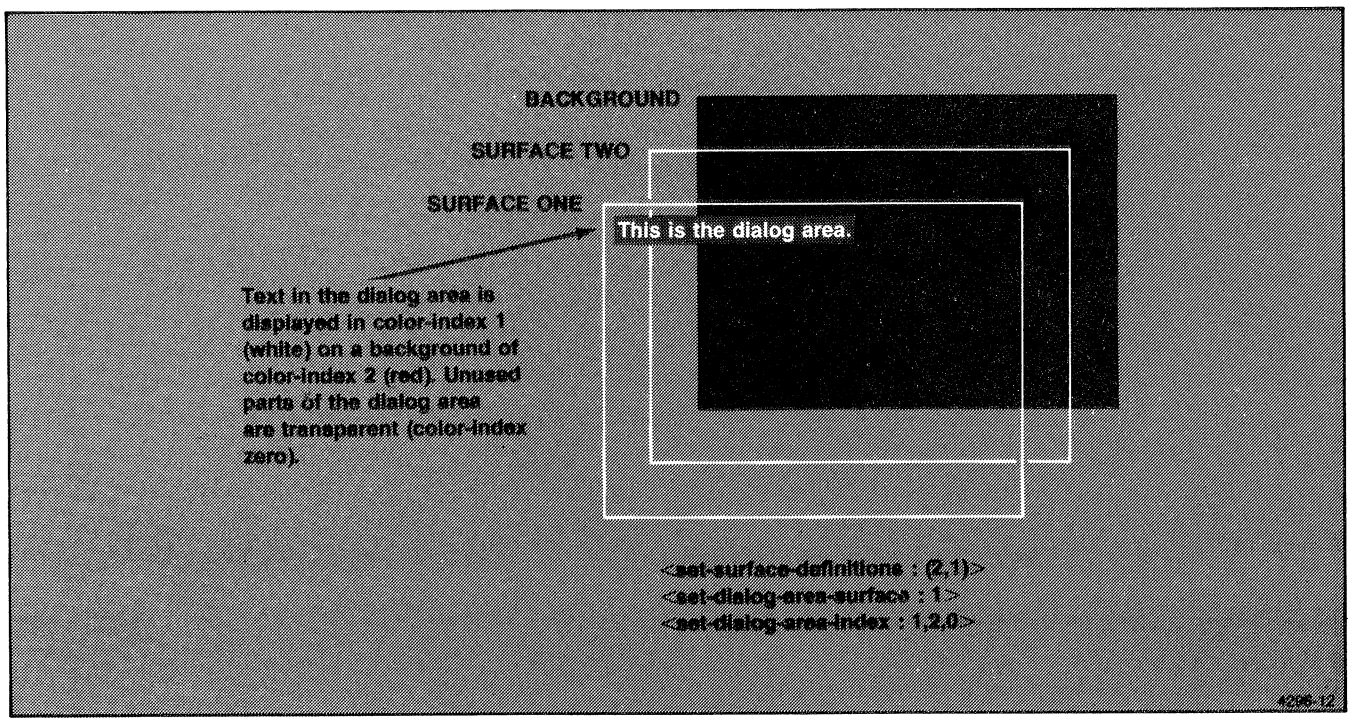


Figure 7-13. Effect of <Set-Dialog-Area-Index> Command.

VIEWS

INTRODUCTION

On command from the host computer, the 4113 can split its display into a number of *viewports*, each of them showing a *view* of different information. For instance, one viewport might show a menu of items for the operator to select, while another viewport shows a graph. Figure 7-14 shows different views displayed in two different viewports on the same screen. (Also shown is a dialog area in its own "dialog viewport.")

Each viewport is like an aperture though which the operator can look at a part of some "picture." Often the entire picture can be seen. However, the operator (or the host computer) can also "zoom in" to show part of the picture in more detail; in that case, other parts will be out of view. (The operator does this with special keyboard keys; see the 4113 Operator's Manual for details. The host computer does this with the <window> command, described later in this section.) Figure 7-15 shows the effect.

SOME DEFINITIONS

This manual uses the word *view* to mean a collection of three associated things:

- A *picture*. A view's picture is defined by a list of all the segments which are visible in that view. Since segments are drawn in 4096-by-4096 terminal space, a view's picture exists in terminal space.
- A *window*. The window is a rectangle in terminal space which bounds the part of the picture which is "in view."
- A *viewport*. The viewport is a rectangle in 640-by-480 raster memory space where the window contents appear. The viewport exists on one of the writing surfaces in raster memory space.

Here, *terminal space* means the 4096-by-4096 space in which segments are defined and moves and draws are performed. *Raster memory space* is the 640-by-480 space in which pixel information is stored for the raster-scan display.

A view's window and viewport define a *window-viewport transform*: a mapping from a subset of terminal space (the window interior) onto a subset of raster memory space (the viewport interior).

NOTE

In the total graphics system, which includes host-resident software as well as the terminal, there may be more than one window-viewport transform. Here we are talking about a window-viewport transform within the terminal, between terminal space (4096-by-4096 space) and raster memory space (640-by-480 space). Do not confuse this window-viewport transform with other window-viewport transforms which may be performed by the host computer software.

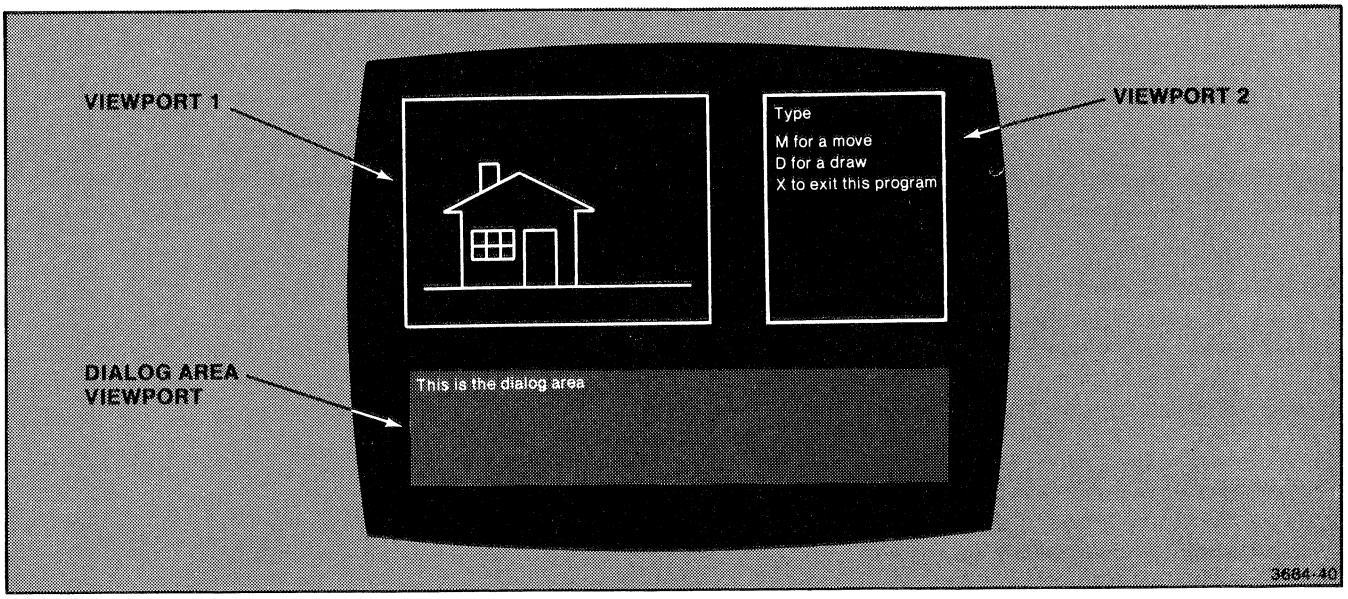


Figure 7-14. Viewports On the 4113 Screen.

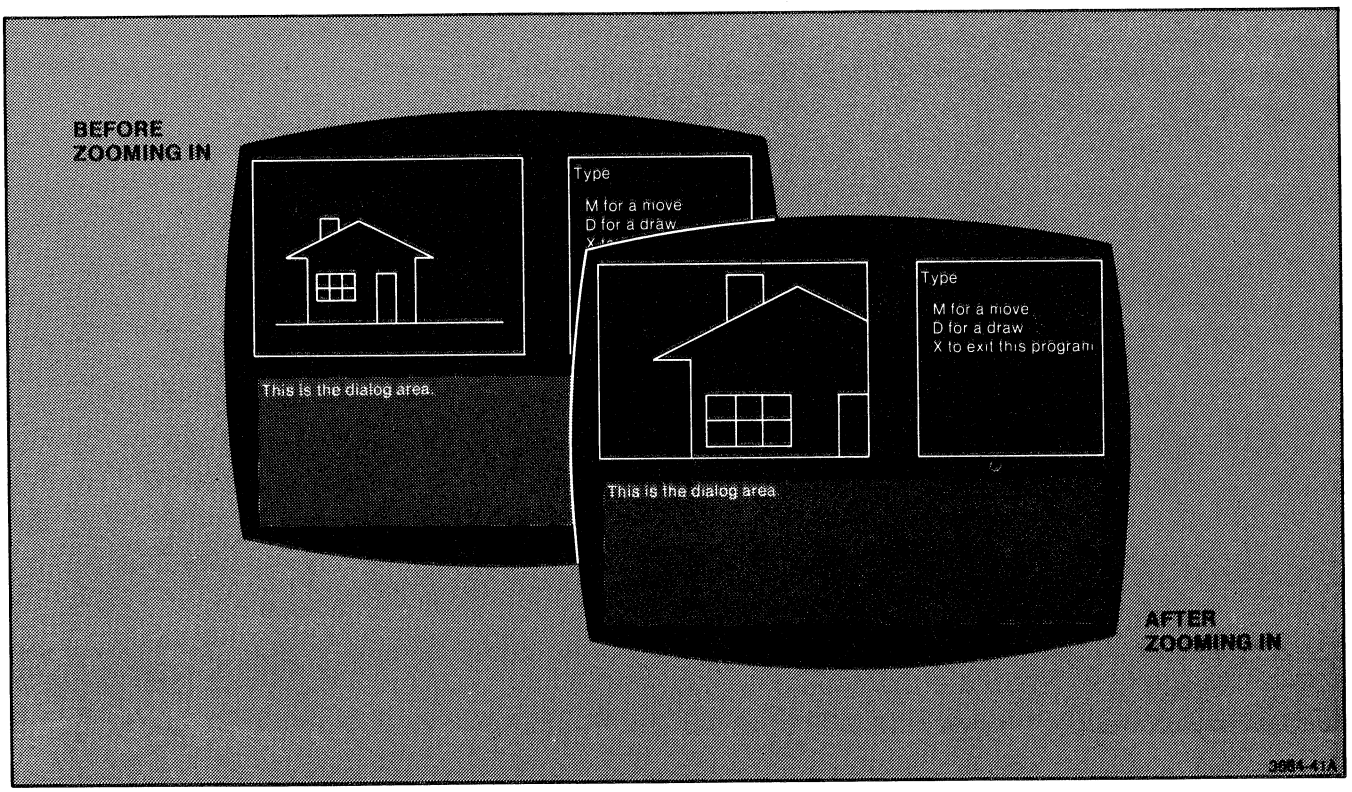


Figure 7-15. Zooming In to See a Part of a Picture in More Detail.

THE DEFAULT VIEW

When the 4113 is first turned on, only one view ("view number 1") is present. In this "default view," the viewport occupies all of surface one in 640-by-480 raster memory space. The window occupies slightly more than three-fourths of terminal space (from X= 0 to X = 4095, and from Y = 0 to Y = 3127). Figure 7-16 shows the default view.

DEFINING OTHER VIEWS

An Example

The following example shows how to define views. (The commands used in this example are all described in the 4110 Series Command Reference Manual.)

Step One. Delete all views which may currently be defined. (The current view then becomes the default view; this default view is "view 1.") Then erase the default view with a <page> command; this has the effect of erasing the screen.

```
<delete-view : all views>
<page>
```

Figure 7-17 shows the effect.

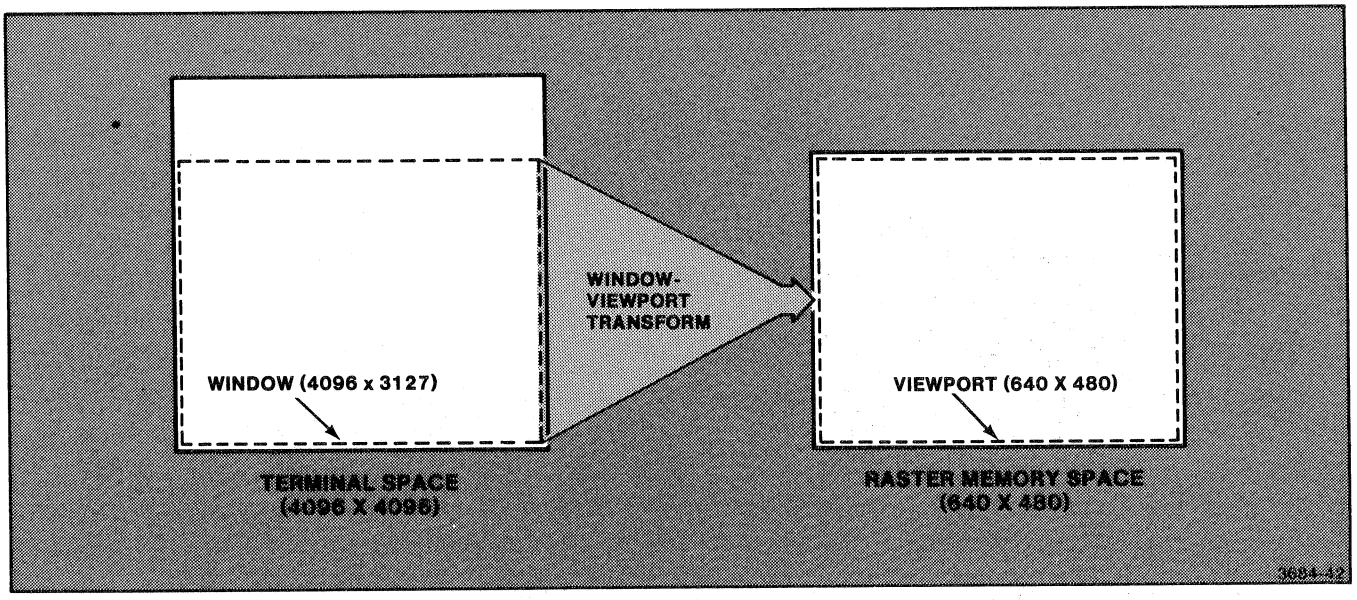


Figure 7-16. The Default View.

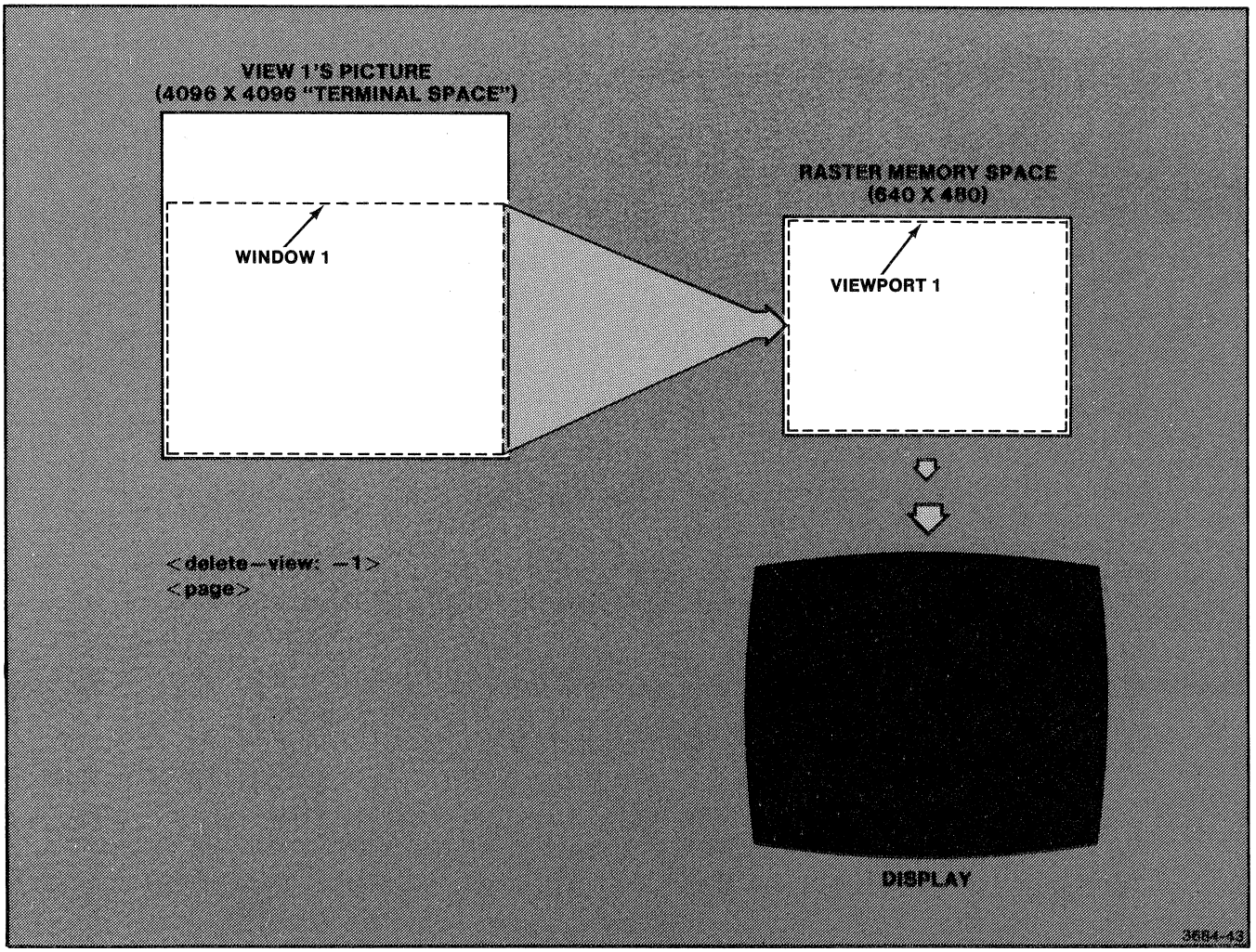


Figure 7-17. Step One: Default View, Empty Screen.

Step Two. Define a new viewport and window for view one. To do this, use the <set-viewport> and <set-window> commands:

<set-viewport> = (ESC)(R)(V)<xy><xy> .

<set-window> = (ESC)(R)(W)<xy><xy> .

In the <set-viewport> command, the two <xy> parameters specify the lower left and upper right corners of the viewport. Likewise, in the <set-window> command, the two parameters specify the lower left and upper right corners of the window.

The window is in the 4096-by-4096 terminal space, so the x- and y-coordinates for its corners can range from 0 to 4095.

The viewport is in the 640-by-480 raster memory space, so you might expect that the <set-viewport> command would require x-coordinates in the range from 0 to 639, and y-coordinates in the range from 0 to 639. However, this could involve tedious calculations to decide which raster memory space coordinates to issue in the <set-viewport> command. To relieve you of these calculations, *the <set-viewport> command accepts its lower-left and upper-right coordinates as if those coordinates were in terminal space rather than raster memory space.*

More precisely, the <set-viewport> command's <xy> parameters are specified in 4096-by-3072 "normalized screen coordinates." That is, the x-coordinates may range from 0 to 4095; y-coordinates may range from 0 to 3071. The 4113 automatically converts these coordinates to the appropriate 640-by-480 raster memory space coordinates.

In this example, the viewport for view one extends (in normalized screen coordinates) from X= 200 to X= 2200, and from Y= 1500 to Y= 3000. The window extends (in 4096-by-4096 terminal space) from X= 0 to X= 4095, and from Y= 0 to Y= 3071. Also, this example uses the <set-border-visibility> command to display a border around the viewport for view one. The commands for this are as follows:

```
<set-viewport : (200,1500),(2200,3000)>
      = (ESC)(R)(V) <xy : (200,1500)>
        <xy : (2200,3000)>

<set-window : (0,0), (4095,3071)>
      = (ESC)(R)(W) <xy : (0,0)>
        <xy : (4095,3071)>

<set-border-visibility : 1>
      = (ESC)(R)(E)<int : 1>
      = (ESC)(R)(E)(1)
```

Figure 7-18 shows the result.

For more information on the <set-window> and <set-viewport> commands, see their descriptions in the 4110 Series Command Reference Manual.

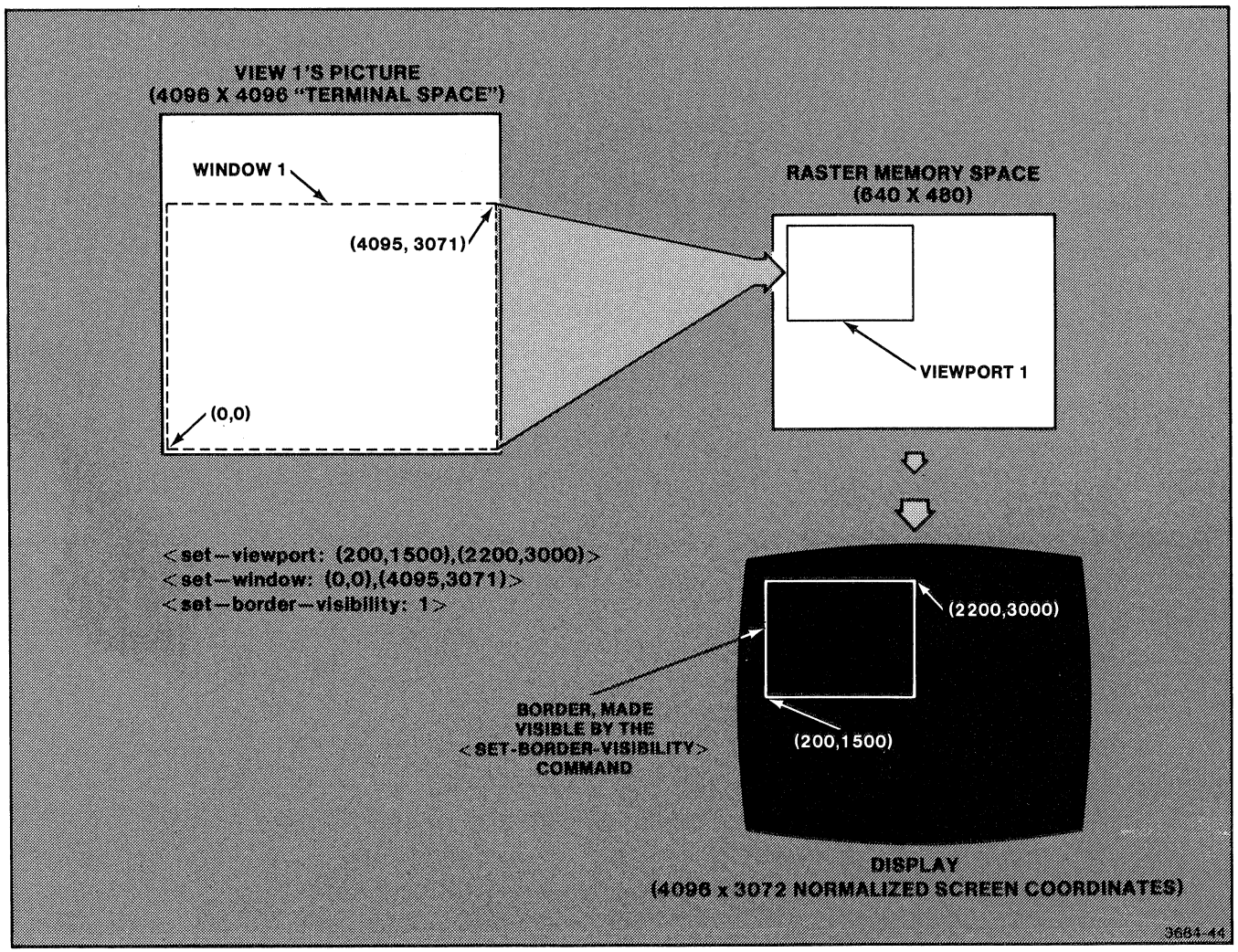


Figure 7-18. Step Two: Setting View One's Viewport and Window.

Step Three. Define the window and viewport for view 2. This is done as follows:

1. Issue a <select-view : 2> command. This automatically brings view 2 into existence, with the same window and viewport as for view 1.

   <select-view : 2>

   = (ESC)(R)(C)
   <int: 2>

   = (ESC)(R)(C)(2) .

2. Define the viewport for view 2, using the <set-viewport> command.

   <set-viewport : (2500,1500), (3900,3000)>

   = (ESC)(R)(V)
   <xy: (2500,1500)>
   <xy: (3900,3000)> .

3. Define the window for view 2. To preserve the same *aspect ratio* (ratio of height to width) as in the viewport, you can use a special feature of the <set-window> command. If you specify a window width (or height) of zero, the terminal computes for you a window width (or height) of the right size so as to preserve the same aspect ratio as was used in the viewport. (For more information on this feature, see the <set-window> description in the 4110 Series Command Reference Manual.)

   The following command chooses a "width" of zero (since the lower-left and upper-right x-coordinates are equal). In executing the command, the 4113 calculates a window width such that the window will have the same aspect ratio as the viewport.

   <set-window : (0,0), (0,3071)>

   = (ESC)(R)(W)
   <xy: (0,0)>
   <xy: (0,3071)> .

4. Finally, issue a <set-border-visibility> command to draw a window around the current viewport:

   <set-border-visibility : 1>

   = (ESC)(R)(E) <int : 1>

   = (ESC)(R)(E)(1) .

Figure 7-19 shows the effect.

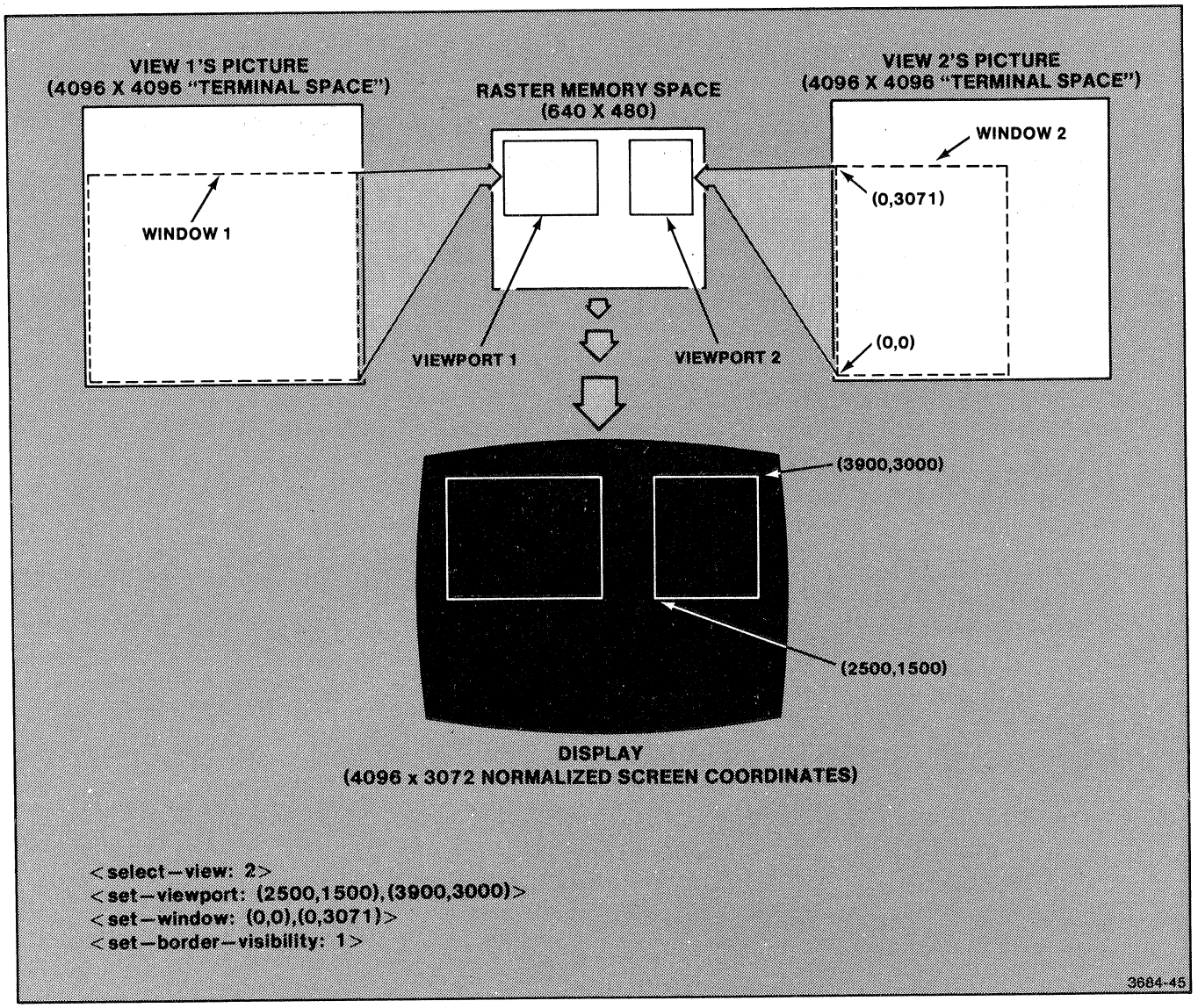


Figure 7-19. Step Three: Defining View Two.

Step Four. Display an alphatext menu in viewport 2. Do this as follows:

1. Disable the dialog area, by issuing an <enable-dialog-area : 0> command. This permits alphatext to go to the current viewport rather than the dialog area.

2. Issue a <page> command. Since view 2 is the current view, and since the dialog area is disabled, the <page> command does the following:

   (a) Erases viewport 2.

   (b) Positions the alpha cursor (for alpha mode text) at (0,3071) in terminal space.

   (c) Puts the terminal in alpha mode, and moves the alpha cursor downward by the height of one alphatext character.

   (For other effects of the <page> command, see its description in the 4110 Series Command Reference Manual.)

3. Send the alphatext menu to the terminal. To prevent accidental erasure of the menu, it is wise to make the menu part of a segment:

```
<delete-segment: 999>
<begin-segment: 999>
<alphatext: "Type M to move, D to draw,
            X to exit.">
<end-segment>
```

4. Enable the dialog area again, so that subsequent alphatext is directed to the dialog area scroll rather than the current viewport.

Figure 7-20 shows the result.

Step Five. Select view 1, and draw a picture there. *Be sure to include the picture within a segment.* (In order for the operator to be able to zoom in on the picture, that picture must be comprised of one or more retained segments.)

```
<select-view : 1>
<delete-segment : 1>
<begin-segment : 1>
   <enter-vector-mode>
      <xy>
      <xy>
      .
      .
      .
   <enter-vector-mode>
      <xy>
      <xy>
      .
      .
      .
<end-segment>
```

Figure 7-21 shows the result.

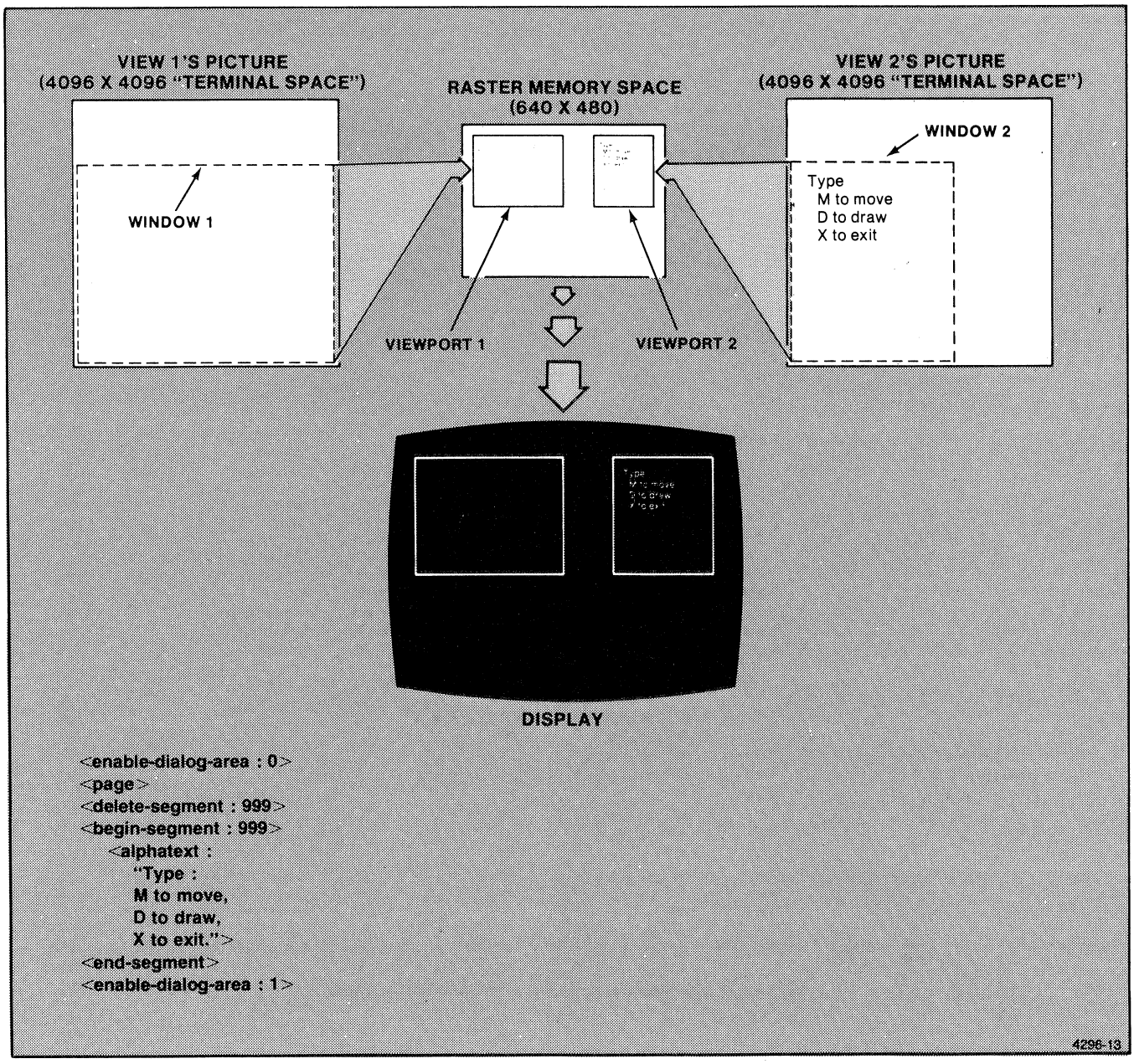


Figure 7-20. Step Four: Putting a Menu Into View Two.

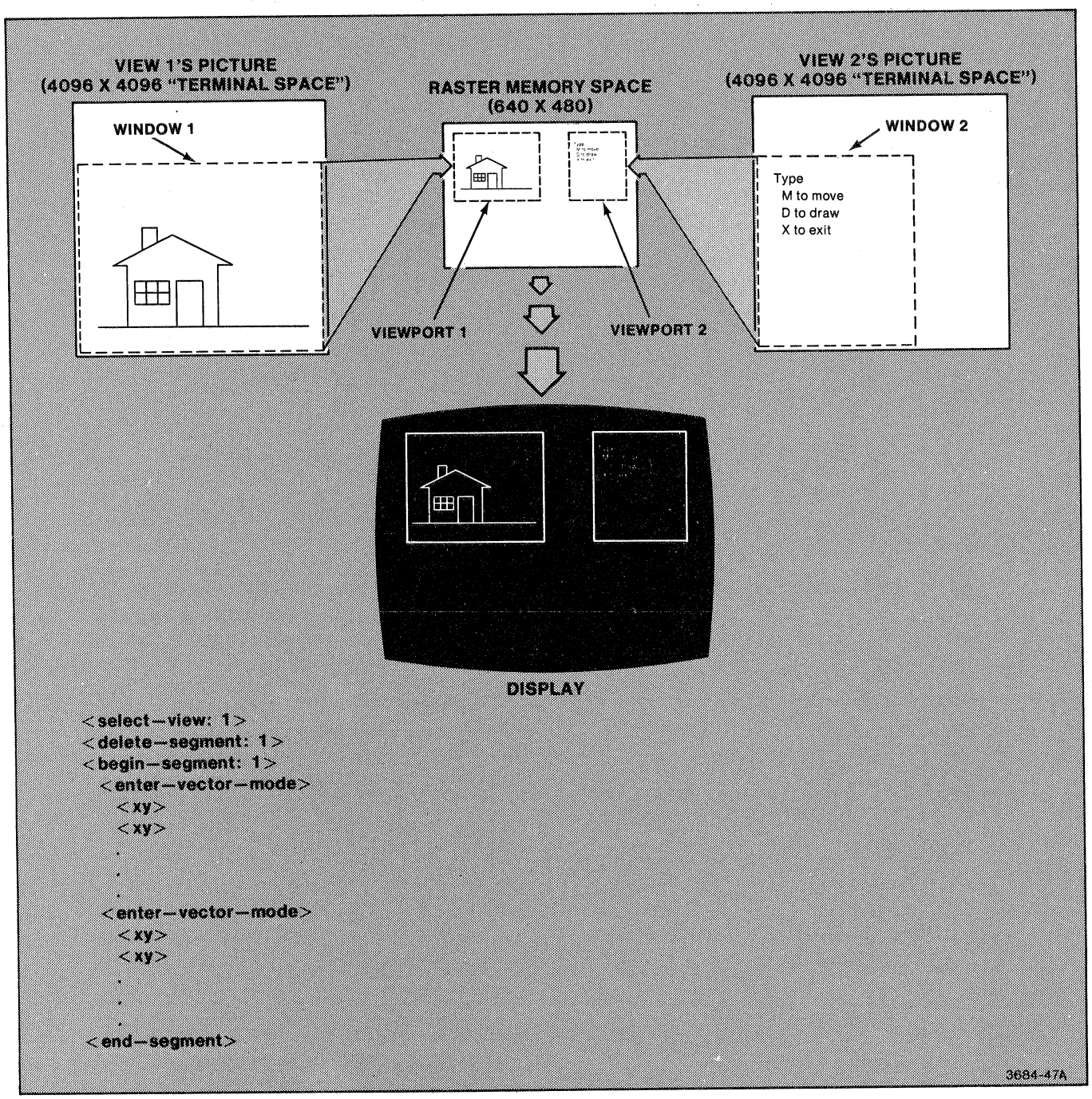


Figure 7-21. Step Five: Putting Graphics Into View One.

Step Six. Define a dialog area. Then enable the dialog area, so that alphatext is displayed there, rather than in the current viewport:

```
<set-dialog-area-chars: 72>
<set-dialog-area-lines: 9>
<set-dialog-area-buffer: 100>
<set-dialog-area-position: (31,31)
<set-dialog-area-index : 1, 3, 3>
<set-dialog-area-visibility : 1>
<enable-dialog-area>
<enter-alpha-mode>
<alphatext : "This is the dialog area.">
```

Figure 7-22 shows the result.

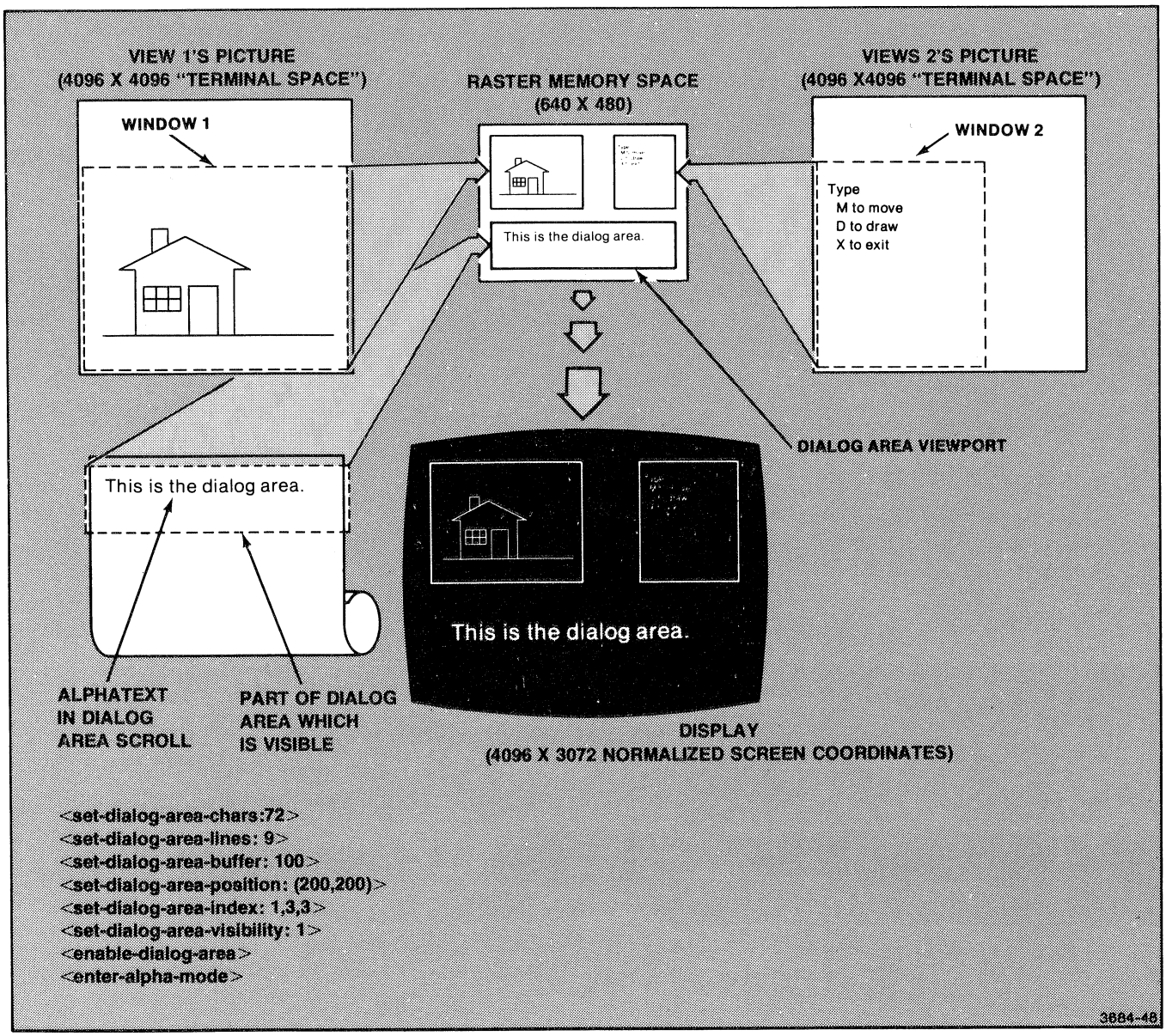


Figure 7-22. Step Six: Creating a Dialog Area.

TYPES OF VIEWPORTS

Besides the numbered viewports associated with the 64 numbered views, the 4113 has two other special viewports: the dialog area viewport, in which the dialog scroll is displayed, and the pixel viewport, in which pixel operations are performed.

The dialog viewport has already been described. (See "The Dialog Area" in Section 4, and "More About the Dialog Area," earlier in this section.) The pixel viewport will be described later in this section, under "Pixel Operations."

ATTRIBUTES OF NUMBERED VIEWPORTS

The attributes of a numbered view's viewport are:

- Its location in raster memory space. This is set by naming two opposite corners of the viewport in a <set-viewport> command.
- The surface on which the viewport is displayed. This is set by the first parameter in a <set-view-attributes> command.
- The "wipe index" for the viewport. This is the color-index to which the viewport pixels are set when the viewport is erased. The wipe index is set by the second parameter in a <set-view-attributes> command.
- The "border index" for the viewport. This is the color-index used for displaying the view's border. The border index is set by the third parameter in a <set-view-attributes> command.

<SET-VIEW-ATTRIBUTES> COMMAND

The <set-view-attributes> command, which sets three of these viewport attributes, has the following syntax:

<set-view-attributes> = (ESC)(R)(A)
<int : *surface-number*>
<int : *wipe-index*>
<int : *border-index*>

PLACING A VIEWPORT ON ANOTHER SURFACE

In the examples so far, the numbered viewports have always been on Surface 1. You can, however, place a viewport on another surface. For instance, you could reserve Surface 1 for the dialog area, and put graphics in a viewport which occupies all of Surface 2:

<set-surface-definitions : (1,2)>	Surface 1 has one bit plane, while Surface 2 has two bit planes.
<delete-view : −1>	Delete all views. The current view (view one) is now the default view, with a viewport which occupies all of Surface 1.
<set-view-attributes : 2, 0, 3>	The current view's viewport is moved to Surface 2. The wipe index is zero (transparent). The border index is three (whose default color is green.)
<set-dialog-area-surface : 1> <set-dialog-area-lines : 33> <set-dialog-area-chars : 80> <set-dialog-area-index : 1, 0, 0> <enable-dialog-area : 1> <set-dialog-area-visibility : 1>	The dialog area's viewport is placed on Surface 1. It can display 33 lines of 80 characters.

DOUBLE BUFFERING WITH ZERO-BIT-PLANE SURFACES

The <set-surface-definitions> command lets you define an imaginary surface which has zero bit planes assigned to it. You can use this feature for double buffering. Consider, for example, the following commands:

<set-surface-definitions : (1,2,0)>	Create Surface 1 with one bit plane, Surface 2 with two bit planes, and Surface 3 with zero bit planes.
<set-dialog-area-surface : 1>	Put the dialog area viewport on Surface 1.
<delete-view : −1>	Delete all views. The current view is view one, with a viewport encompassing all of Surface 1.
<set-view-attributes : 2, 0, 3>	Move view one's viewport to Surface 2. The wipe index for that viewport is zero; the border index is color-index three.
<select-view : 2>	Create view two. Initially view two's viewport is the same as view one's.
<set-view-attributes : 3, 0, 3>	Put view two's viewport on Surface 3. (It cannot be displayed —for the moment— because Surface 3 has no bit planes assigned to it.) The wipe index and border index are the same as before.
<select-view : 1>	Select view one as the current view.
<begin-segment : 1> <enter-vector-mode> <xy> <xy> . . . <end-segment>	Draw a picture in view one. (The picture in view one is displayed on Surface 2, because view one's viewport is on that surface.)
<select-view : 2> <begin-segment : 2> <enter-vector-mode> <xy> <xy> . . . <end-segment>	Draw a picture in view two. (Since view two's viewport is on Surface 3, and there are no bit planes assigned to Surface 3, the terminal's operator cannot see this picture as it is being built.)

At this point, the situation is as shown in Figure 7-23. The dialog area is on Surface 1. The 4113 has stored the picture for view one in its terminal space memory; it displays that picture in viewport one on Surface 2. The picture for view two is also stored in the 4113's terminal space memory. However, view two cannot be displayed, because its viewport is on Surface 3, which has no bit planes assigned.

Next, the host program issues another <set-surface-definitions> command, so that Surface 3 is assigned two bit planes. Surface 2 now has zero bit planes. When view two is renewed, the operator will see an image of view two. View one is now out of sight, because its viewport is on a surface which has zero bit planes:

<set-surface-definitions : (1,0,2)> Assign two bit planes to Surface 3. This permits view two, whose viewport is on Surface 3, to be displayed.

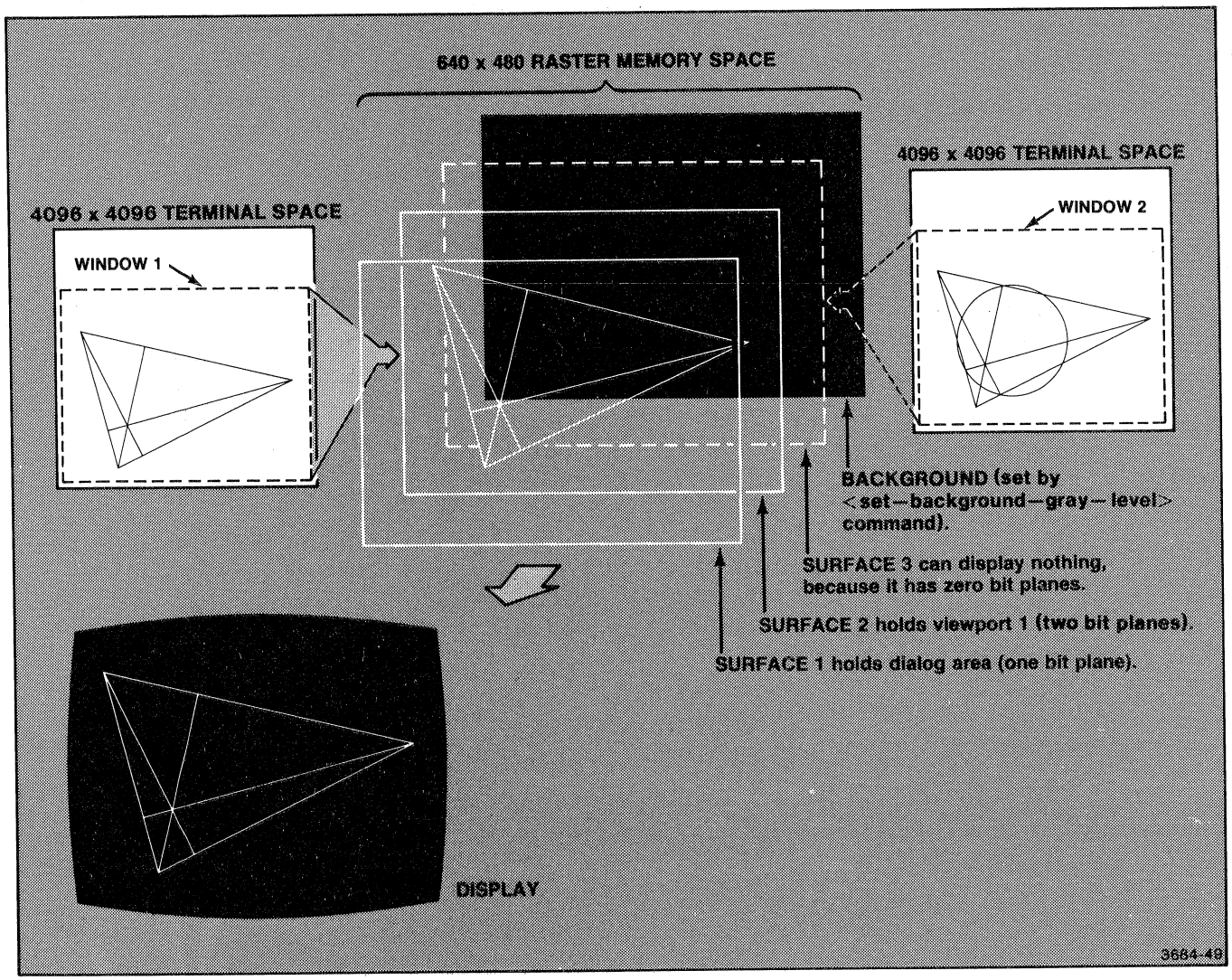


Figure 7-23. Double Buffering: Before Redefining the Surfaces.

<select-view : 2> <renew-view : 0>	The <select-view> command selects view two as the current view. Then the <renew-view : 0> command renews the current view. This causes the view two's picture (in terminal space) to be mapped onto viewport two (on Surface 3 in raster memory space).	<select-view : 1> <delete-segment : 1> <begin-segment : 1> <enter-vector-mode> <xy> <xy> . . . <end-segment>	Update view one's picture in terminal space.

Figure 7-24 shows the result of these commands.

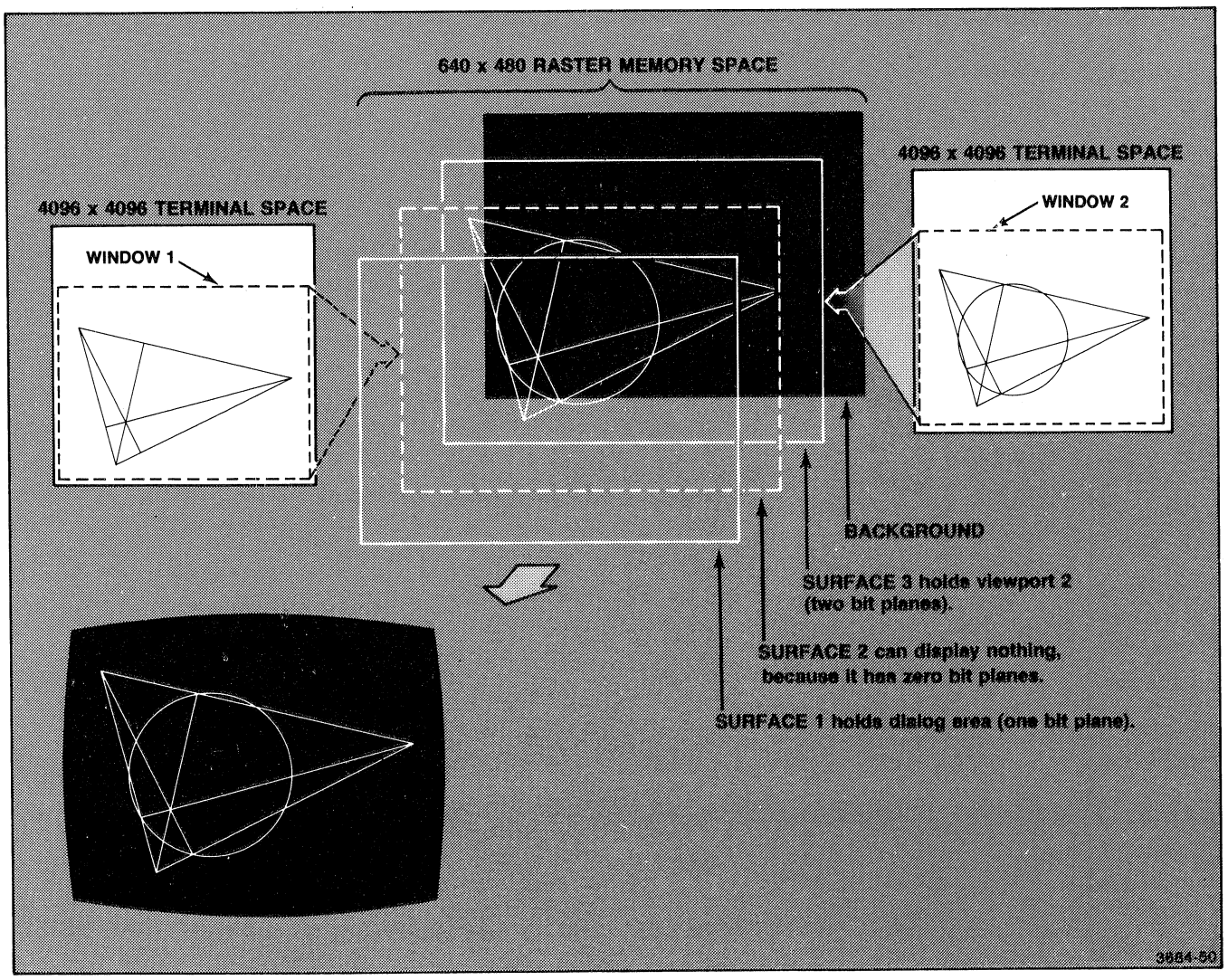


Figure 7-24. Double Buffering: After Redefining the Surfaces.

VIEW DISPLAY CLUSTERS

In the 4113 terminal, you can group several views together in a "view display cluster." This causes all views in the cluster to have identical windows for their window-viewport transforms. That is, changing the window for any view in a view display cluster also changes the windows for all other views in the cluster. Also, renewing any view of the cluster (redisplaying its viewport contents) also renews all other views in the cluster.

This is especially useful when you have several views on different surfaces, describing different aspects of the same object. For instance, suppose you are using the terminal as a light table for preparing multi-layered etched circuit board layouts. In that case, views one, two, three, and four might have identical viewports on Surfaces 1, 2, 3, and 4, respectively. View one would represent the top layer of a circuit board, while views two, three, and four would represent other layers of the same circuit board. In such an application, you would want the framing keys, and the <set-window> and <page> commands, to affect all four views. That way, when you use the framing keys to zoom in on a part of the picture, the zoom operation affects all four views, on all four surfaces.

<SET-VIEW-DISPLAY-CLUSTER> COMMAND

The <set-view-display-cluster> command groups views into a view display cluster. This command has the following syntax:

<set-view-display-cluster>

= (ESC)(R)(Q)
<int-array: *view-numbers*> .

For instance, to group views one, two, three, and four into a view display cluster, you would issue the following command:

<set-view-display-cluster: (1,2,3,4)>

= (ESC)(R)(Q)
<int-array: (1,2,3,4)>

= (ESC)(R)(Q)(4)(1)(2)(3)(4) .

Thereafter, whenever you change the window for any one of these four views, the windows for the other three views also change. This happens whether you change the window explicitly, with the <set-window> command, or implicitly, with the VIEW, CTRL-VIEW, RESTORE, CTRL-RESTORE, OVERVIEW, or CTRL-OVERVIEW keys.

Likewise, whenever you renew any of these four views, all the other views in the cluster are also renewed. This happens with the <renew-view> and <page> commands, and with the PAGE, VIEW, CTRL-VIEW, RESTORE, CTRL-RESTORE, OVERVIEW, and CTRL-OVERVIEW keys.

A view cannot belong to more than one display cluster. Thus, including a view in one cluster automatically removes it from any other clusters.

AN EXAMPLE

Figure 7-25 shows the terminal's screen as it might appear while the host is running an applications program for circuit board layout. The terminal has three surfaces defined, of one bit plane each. On Surface 1, color-index one is white; on Surface 2, index one is red; and on Surface 3, index one is green. Viewports one, two, and three are superimposed above each other on Surfaces 1, 2, and 3, respectively.

In the corner of the screen, viewports four, five, and six contain copies of the information in views one, two, and three respectively. These viewports are also on Surfaces 1, 2, and 3; however, they are staggered so as to show the circuit board overlays in perspective.

Another viewport holds reminders for the operator, while the dialog viewport displays prompt messages and other dialog between the operator and the host computer.

In this application, views one through six are grouped together in view display cluster. Thus, if the operator zooms in to see one view in more detail, the zoom operation affects all views in the display cluster.

For more information on the <set-view-display-cluster> command, see its description in the 4110 Series Command Reference Manual.

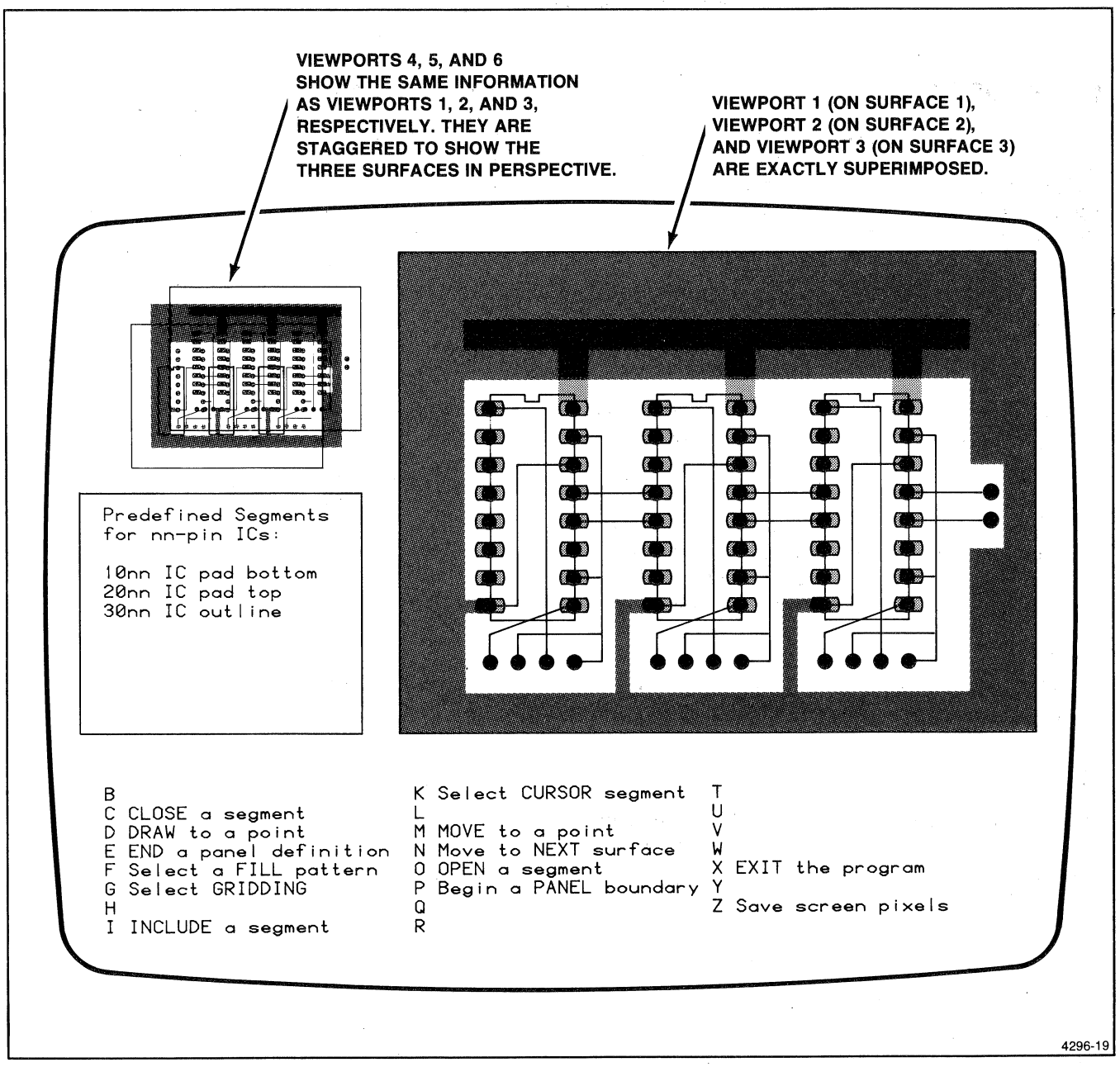


Figure 7-25. Several Views in the Same View Display Cluster.

DEFINING YOUR OWN FILL PATTERNS

INTRODUCTION

A fill pattern is a rectangular array of color-indices used to fill panels. You can use the predefined fill patterns (patterns 1 through 16 and 0 through −15), or you can define your own fill patterns. The pattern definition process is as follows:

1. Issue a <begin-fill-pattern> command.
2. Specify, with <raster-write> or <runlength-write> commands, the color-indices of the individual pixels which make up the fill pattern.
3. Issue an <end-fill-pattern> command. (In certain circumstances, this command is not needed.)

In other words, a fill pattern definition has the following syntax:

<fill-pattern-definition> = <begin-fill-pattern>
[<pixel-def> . . .]
<end-fill-pattern>

where

<pixel-def> = <raster-write> or <runlength-write> .

Example

To define the fill pattern in Figure 7-26 as fill pattern number 60, you can issue the following commands to the terminal:

```
<begin-fill-pattern: 60, 4, 4, 6>
  <raster-write : 8, "77007700">
  <raster-write : 8, "00770077">
<end-fill-pattern>
```

The <raster-write> command is described briefly here, and in more detail later in this section.

<Begin-Fill-Pattern> Command. The <begin-fill-pattern> command's first parameter specifies that pattern 60 is to be defined. The second and third parameters set the fill pattern's width and height to four pixels each. The fourth parameter sets the "bits-per-pixel" value used in decoding the <raster-write> commands that follow. With "bits-per-pixel" set to 6, each color-index can be sent using the corresponding ASCII character: (0) for color-index 0, (7) for color-index 7.

<begin-fill-pattern>

= (ESC)(M)(D)
<int: *pattern-number*>
<int: *pattern-width*>
<int: *pattern-height*>
<int: *bits-per-pixel*>

= (ESC)(M)(D)
<int: 60>
<int: 4>
<int: 4>
<int: 6>

= (ESC)(M)(D)
(C)(<)
(4)
(4)
(6)

= (ESC)(M)(D)(C)(<)(4)(4)(6) .

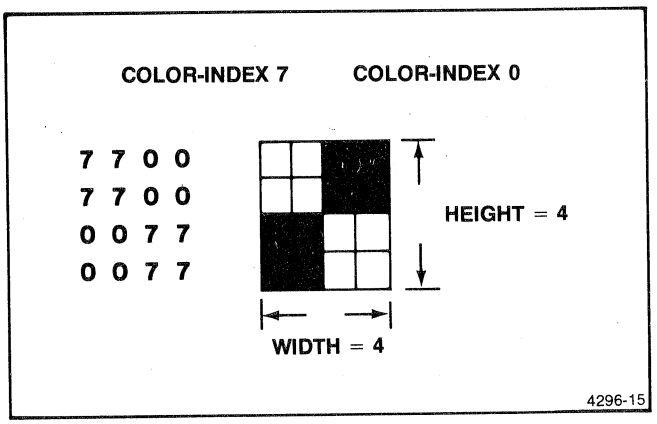


Figure 7-26. A Simple Fill Pattern.

<Raster-Write> Command. The two <raster-write> commands specify the color-indices of individual pixels in the fill pattern. With "bits-per-pixel" set to six, the individual characters in the <char-array> parameters can be (0) to represent color-index 0, (1) for color-index one, etc.

<raster-write: 8, "77007700">

= (ESC)(R)(P)
<int: 8>
<string: "77007700">

= (ESC)(R)(P)
(8)
(8)(7)(7)(0)(0)(7)(7)(0)(0)

= (ESC)(R)(P)(8)(8)(7)(7)(0)(0)(7)(7)(0)(0) .

<raster-write: 8, "00770077">

= (ESC)(R)(P)
<int: 8>
<string: "00770077">

= (ESC)(R)(P)
(8)
(8)(0)(0)(7)(7)(0)(0)(7)(7)

= (ESC)(R)(P)(8)(8)(0)(0)(7)(7)(0)(0)(7)(7) .

The first parameter in each <raster-write> command is an <int> telling how many pixels have their color-indices specified in that command. The second parameter is a <string>: an array of individual <char>s. (Another word for "<string>" is "<char-array>.") In the <raster-write> command, the individual <char>s in the <char-array> must be in the range from (SP) to ('); that is, their decimal equivalents must be in the range from 32 to 96. With bits-per-pixel set to six, each <char> is a code telling the color-index for one pixel.

(If bits-per-pixel were set to 1, 2, 3, or 4, then each <char> would hold the color-indices for 6, 3, 2, or 1.5 pixels, respectively. For details, see the description of the <raster-write> command in the 4110 Series Command Reference Manual.)

The pattern definition begins with the upper left pixel of the fill pattern and proceeds from left to right, with wrap-around as required.

Ending a Fill Pattern Definition. The usual way to end a fill pattern definition is with an <end-fill-pattern> command:

<end-fill-pattern> = (ESC)(M)(E) .

However, the pattern definition can also end in two other ways:

- The pattern definition ends when all color-indices in the pattern have been specified.
- If a <begin-pixel-operations> command is received, that command terminates any fill pattern definition which may be in progress. (The <begin-pixel-operations> command is described later in this section under "Pixel Operations.")

In the previous example, the two <raster-write> commands together specified all 16 color-indices in the fill pattern. Thus, the <end-fill-pattern> command was not really needed. However, including this command does no harm. Including this command guarantees that the fill pattern definition does terminate, even if you make a mistake and do not send all the necessary color-indices in the <raster-write> commands.

MORE ABOUT THE <RASTER-WRITE> COMMAND

The <raster-write> command is used to send each color-index separately. The command has the following syntax:

<raster-write> = (ESC)(R)(P)
<int: *number-of-pixels*>
<char-array: *codes-holding-color-indices*> .

If the preceding <begin-fill-pattern>'s command specified six bits per pixel, then the terminal only pays attention to the least-significant four bits of the code characters in the <char-array>. This lets you use the character (0) to represent color-index 0, the character (1) to represent color-index 1, and so on.

If the <begin-fill-pattern> command specified one, two, three, or four bits per pixel, then the code characters are composed as follows:

1. The color-indices to be placed in the fill pattern are regarded as one-bit, two-bit, three-bit, or four-bit binary numerals. For instance, if bits-per-pixel = 2, then the color-indices 0, 0, 3, 2, 1, 0 are represented as follows:

   00 00 11 10 01 00

2. The binary numerals are grouped together to form six-bit binary numerals. For instance the color-indices 0, 0, 3, 2, 1, 0 are represented as follows:

   000011 100100

3. An offset of 32 (binary 100000) is added to each binary numeral. The resulting binary numeral is used as a code character in the <char-array> parameter. (These characters are in the range from (SP) to (_), and have decimal equivalents in the range from 32 to 95.) For instance, the gray-indices 0, 0, 3, 2, 1, 0 are represented as follows:

   (#)(D)

4. The code characters are packed into the <raster-write> command's <string> or <char-array> parameter. The number of pixels whose color-indices are represented in those character is given in the <int> parameter. Thus, with two bits-per-pixel, the color-indices 0, 0, 3, 2, 1, 0 are represented in a <raster-write> command as follows:

<raster-write> = (ESC)(R)(P)
<int: 6>
<char-array: "#D">

= (ESC)(R)(P)(6)(2)(#)(D) .

Example: Six Bits Per Pixel

Consider the fill pattern in Figure 7-27.

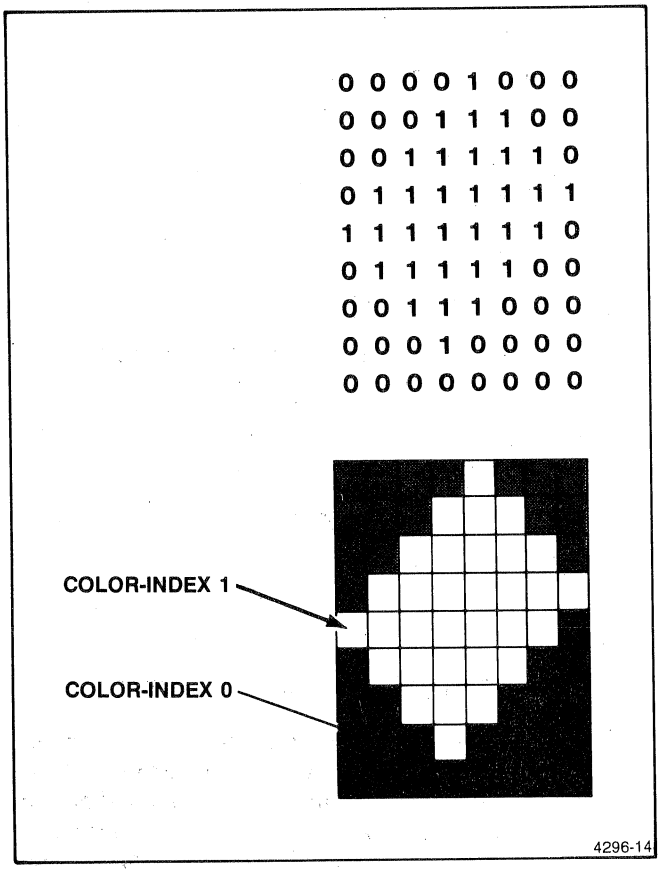


Figure 7-27. A Fill Pattern.

Using <raster-write> commands with six bits per pixel, the fill pattern can be defined as follows:

```
<begin-fill-pattern: 20, 8, 9, 6>
   <raster-write: 8, "00001000">
   <raster-write: 8, "00011100">
   <raster-write: 8, "00111110">
   <raster-write: 8, "01111111">
   <raster-write: 8, "11111110">
   <raster-write: 8, "01111100">
   <raster-write: 8, "00111000">
   <raster-write: 8, "00010000">
   <raster-write: 8, "00000000">
<end-fill-pattern>
```

The first two of these commands expand into individual ASCII characters as follows:

<begin-fill-pattern: 20, 8, 9, 6>

= (ESC)(M)(D)
<int: 20>
<int: 8>
<int: 9>
<int: 6>

= (ESC)(M)(D)
(A)(4)
(8)
(9)
(6)

= (ESC)(M)(D)(A)(4)(8)(9)(6) .

<raster-write: 8, "00001000">

= (ESC)(R)(P)
<int: 8>
<char-array: "00001000">

= (ESC)(R)(P)
(8)
(8)(0)(0)(0)(0)(1)(0)(0)(0) .

The other commands expand into ASCII characters in a like manner. The result is the following character sequence:

(ESC)(M)(D)(A)(4)(8)(9)(6)
(ESC)(R)(P)(8)(8)(0)(0)(0)(0)(1)(0)(0)(0)
(ESC)(R)(P)(8)(8)(0)(0)(0)(1)(1)(1)(0)(0)
(ESC)(R)(P)(8)(8)(0)(0)(1)(1)(1)(1)(1)(0)
(ESC)(R)(P)(8)(8)(0)(1)(1)(1)(1)(1)(1)(1)
(ESC)(R)(P)(8)(8)(1)(1)(1)(1)(1)(1)(1)(0)
(ESC)(R)(P)(8)(8)(0)(1)(1)(1)(1)(1)(0)(0)
(ESC)(R)(P)(8)(8)(0)(0)(1)(1)(1)(0)(0)(0)
(ESC)(R)(P)(8)(8)(0)(0)(0)(1)(0)(0)(0)(0)
(ESC)(R)(P)(8)(8)(0)(0)(0)(0)(0)(0)(0)(0)
(ESC)(M)(E)

Example: End-Of-Row Character

The <raster-write> command has a special character, ('), which you can include in the <char-array> parameter to mark the end of a row of pixels. This causes the next color-index to go into the first pixel of the following row. The pixels left undefined are set to color-index zero.

By using the end-of-row character, the preceding commands can be shortened, as follows:

```
<begin-fill-pattern: 20, 8, 9, 6>
   <raster-write: 5, " 00001' ">
   <raster-write: 6, " 000111' ">
   <raster-write: 7, " 0011111' ">
   <raster-write: 8, " 01111111 ">
   <raster-write: 7, " 1111111' ">
   <raster-write: 6, " 011111' ">
   <raster-write: 5, " 00111' ">
   <raster-write: 4, " 0001' ">
<end-fill-pattern>
```

(The <end-fill-pattern> command terminates the fill pattern before the last row of pixels have been defined. Consequently, these pixels are set to color-index zero.)

You can combine several rows of pixels into one <raster-write> command, which shortens the <fill-pattern-definition> even more:

```
<begin-fill-pattern: 20, 8, 9, 6>
  <raster-write: 18, " 00001'000111'0011111' ">
  <raster-write: 15, " 011111111111111' ">
  <raster-write: 15, " 011111'00111'0001' ">
<end-fill-pattern>
```

These commands expand to the following character sequences:

```
(ESC)(M)(D)(A)(4)(8)(9)(6)
(ESC)(R)(P)(A)(2)(A)(5)(0)(0)(0)(0)(1)(')
  (0)(0)(0)(1)(1)(1)(')(0)(0)(1)(1)(1)(1)(1)(')
(ESC)(R)(P)(?)(A)(0)(0)(1)(1)(1)(1)(1)(1)
  (1)(1)(1)(1)(1)(1)(1)(1)(')
(ESC)(R)(P)(?)(A)(2)(0)(1)(1)(1)(1)(1)(')
  (0)(0)(1)(1)(1)(')(0)(0)(0)(1)(')
(ESC)(M)(E)
```

Example: One Bit Per Pixel

Since the fill pattern in our example uses only color-indices zero and one, the <raster-write> commands can be sent yet more concisely by setting the number of bits per pixel to one. In that case, each color-index is represented by only one bit, and six color-indices fit into each character of the <char-array>.

The color-indices for the fill pattern are as follows:

```
0 0 0 1 0 0 0 0
0 0 1 1 1 0 0 0
0 1 1 1 1 1 0 0
1 1 1 1 1 1 1 0
0 1 1 1 1 1 1 1
0 0 1 1 1 1 1 0
0 0 0 1 1 1 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 0 0 0
```

These are grouped into six-bit binary numerals, as follows:

```
000100 000011 100001 111100 111111 100111
111100 111110 000111 000000 100000 000000
```

By adding 32 (binary 100000), these are converted to ASCII characters in the range from (SP) to (_). Since the last six-bit group consists only of zeroes, it may be omitted; an <end-fill-pattern> command will set these pixels to color-index zero:

0100100 ($)	0100011 (#)	1000001 (A)	1011100 (\)
1011111 (_)	1000111 (G)	1011100 (\)	1011110 (^)
0100111 (')	0100000 (SP)	1000000 (@)	

Thus, the characters in the <raster-write> command's <char-array> are as follows:

($)(#)(A)(\)(_)(G)(\)(^)(')(SP)(@).

The <fill-pattern-definition> is as follows:

```
<begin-fill-pattern: 20, 8, 9, 1>
    <raster-write: 64, " $#A\_G\^'   @">
<end-fill-pattern>
```

This expands into the following sequence of ASCII characters:

```
(ESC)(M)(D)(A)(4)(8)(9)(1)
(ESC)(R)(P)(D)(0)(;)($)(#)(A)(\)(_)(G)(\)
    (^)(')(SP)(@)
(ESC)(M)(E)
```

Summary

It is easiest, when designing fill patterns, to set "bits-per-pixel" to six and use <raster-write> commands to specify the color-indices for the fill pattern. That way, each color-index is sent as a single ASCII character: (0) for color-index zero, (1) for color-index one, and so on. But while this is easy, it is not concise.

By setting bits-per-pixel to four, you can pack one and one-half color-indices into each character of the <raster-write> command's <char-array> parameter. With four bits per pixel, the color-indices can range from 0 to 15.

By setting bits-per-pixel to three, you can pack two color-indices into each character of the <char-array>. With three bits per pixel, the color-indices can range from 0 to 7.

By setting bits-per-pixel to two, you can pack three color-indices into each character of the <char-array>, and those color-indices can range from 0 to 3.

By setting bits-per-pixel to one, you can pack six color-indices into each character of the <char-array>. However, the only color-indices possible in that case are 0 and 1.

For more information about the <raster-write> command, see its description in the 4110 Series Command Reference Manual.

<RUNLENGTH-WRITE> COMMAND

If you like, you can use the <runlength-write> command instead of the <raster-write> command when defining fill patterns. This command is useful when you have one index value that is repeated many times in sequence (for instance, the value 6 repeated 30 times: 6,6,6, . . . 6,6,6).

The <runlength-write> command has the following syntax:

<runlength-write> = (ESC)(R)(L)
<int-array: *runcode-array*>

Each runcode in the <int-array> is a single number into which are packed two other numbers. The least-significant bits of the runcode hold a color-index. (If bits-per-pixel = N, then these are the N least-significant bits of the runcode.) The most-significant bits hold a "run length," which tells how many times the gray-index is to be repeated.

Let N be the number of bits per pixel, as specified in the <begin-fill-pattern> command. Let L be the length of a run of pixels. Let I be the color-index of each pixel in that run. Then the runcode R is computed as follows:

$$R = (2^N) L + I$$

Example

Table 7-2 shows how the fill pattern of Figure 7-27 may be expressed in runcodes, assuming one bit per pixel.

Table 7-2
EXPRESSING A FILL PATTERN WITH RUNCODES

Fill Pattern	Length of Run (L)	Index (I)	Runcode (2L+I)
00001000	4	0	8
00011100	1	1	3
00111110	6	0	12
01111111	3	1	7
11111110	4	0	8
01111100	5	1	11
00111000	2	0	4
00010000	14	1	29
00000000	2	0	4
	5	1	11
	4	0	8
	3	1	7
	6	0	12
	1	1	3
	12	0	(Omitted)

Referring to Table 7-2 for the runcodes in the <runlength-write> command, the fill pattern can be defined as follows:

<begin-fill-pattern: 20,8,9,1>
<runlength-write: (8,3,12,7,8,11,4,29,4,11,8,7,12,3)>
<end-fill-pattern>

These commands expand into the following ASCII characters:

(ESC)(M)(D)(A)(4)(8)(9)(1)
(ESC)(R)(L)(>)(8)(3)(<)(7)(8)(;)(4)(A)(=)(4)(;)
(8)(7)(<)(3)
(ESC)(M)(E)

OTHER CONSIDERATIONS

Editing Fill Patterns

Once a fill pattern has been defined, it may not have individual areas changed ("edited"), nor can it be added to. The only way to change an existing fill pattern is to redefine it.

Deleting Fill Patterns

There is no "delete fill pattern" command. To delete an existing fill pattern, you can redefine it with a height of zero. For instance, to delete fill pattern 20, issue a <define-fill-pattern> command with a height of zero:

<define-fill-pattern: 20, 1, 0, 1>

Surfaces With Too Few Bit Planes

If a fill pattern which was defined with N bits per pixel is displayed on a surface with fewer than N bit planes, then not all possible color-indices in the pattern definition can be displayed.

For instance, suppose that when a pattern is defined, the bits-per-pixel parameter in the <begin-fill-pattern> command is 3. In that case, there are eight possible color-indices in the pattern: color-index 0 through color-index 7. Suppose further that the pattern is displayed on a surface which has only two bit planes. In that case, the only color-indices which can be displayed are those numbered from 0 to 3.

In such a situation, the terminal uses only the most-significant bits in each color-index of the fill pattern definition. Thus, to display a pattern defined with three bits per pixel, on a two-bit-plane surface, the terminal displays indices 7 and 6 as color-index 3. Likewise, it displays indices 5 and 4 as index 2; it displays indices 3 and 2 as index 1; and it displays indices 1 and 0 as index 0.

The predefined fill patterns (patterns −15 to + 16) were defined (at the Tektronix factory) using the minimum possible number of bits per pixel. For instance, patterns −1, 0, 1, 2, . . . , 14, and 16 each contain only the color-indices 0 and 1. Therefore, these patterns were defined with only one bit per pixel. Patterns −2 and −3 require two bits per pixel, since they contain color indices 2 (binary 10) and 3 (binary 11). Again, patterns −4 to −7 were defined with three bits per pixel. That is because they contain color-indices 4, 5, 6, and 7 (binary 100, 101, 110, and 111), which require three binary bits. The only predefined fill patterns with four bits per pixel are patterns −8 through −15 and pattern + 15.

PIXEL OPERATIONS

INTRODUCTION

With the commands described so far, when you want to draw a line you specify the end points of that line in 4096-by-4096 terminal space. The 4113 terminal then automatically computes the pixels through which the line passes, and displays those pixels in the appropriate color-index. You do not have to specify the individual pixels yourself, nor do you have to write a host program to do that for you. The 4113's internal processor takes care of those details.

However, if you wish, you can access the individual pixels of the terminal's raster memory buffer. To let you do this, the terminal has a number of "pixel commands."

NOTE

The pixel commands affect only the raster memory buffer (the bit planes). They do not modify any segments in terminal space, nor do they have any effect on the numbered views and their window-viewport transforms.

You can use pixel commands to write over the viewport for the current view. However, when the current view is renewed (by the <page> command, PAGE key, <renew-view> command, etc.), the effect of the pixel commands on that viewport is lost. (This is because the pixel commands affect only the raster memory buffer, and are not stored in a segment.)

The pixel commands fall into three categories: "preparation commands," "pixel writing commands," and the <save> command.

Preparation Commands

There are three preparation commands: <begin-pixel-operations>, <set-pixel-viewport>, and <set-pixel-beam-position>.

<Begin-Pixel-Operations>. The <begin-pixel-operations> command specifies the "pixel writing surface"—the surface whose pixels are to be modified by subsequent <raster-write> and <runlength-write> commands. This command also sets two environmental parameters used by subsequent <raster-write> and <runlength-write> commands. (These environmental parameters are "ALU mode" and "bits-per-pixel." They are described later in this section, and in the 4110 Series Command Reference Manual.)

<Set-Pixel-Viewport>. The <set-pixel-viewport> command defines a rectangular region on the pixel writing surface. Subsequent <raster-write> and <runlength-write> commands affect only that rectangular region.

<Set-Pixel-Beam-Position>. The <set-pixel-beam-position> command specifies the individual pixel (in the pixel viewport) where the next <raster-write> or <runlength-write> command will start changing the color-indices of individual pixels.

Pixel Writing Commands

The pixel writing commands write color-indices into pixels of the raster memory buffer. Two of these, <raster-write> and <runlength-write>, give you control over the individual pixels in the pixel viewport. (These commands have already been described in connection with fill pattern definitions, earlier in this section.)

The other pixel writing commands are <pixel-copy> and <rectangle-fill>. The <pixel-copy> command copies pixels from one rectangular region in raster memory space to another rectangular region. The <rectangle-fill> command sets all pixels within a rectangular region to the same color-index.

The <Save> Command

The <save> command lets you save some or all of the pixels in the current pixel viewport. The pixels are saved on a disk file (or other output device) as a series of <runlength-write> or <raster-write> commands.

References

For detailed information on the individual pixel operations commands, see these descriptions in the 4110 Series Command Reference Manual:

<begin-pixel-operations>
<pixel-copy>
<raster-write>
<rectangle-fill>
<runlength-write>
<set-pixel-beam-position>
<set-pixel-viewport>

The rest of this section is devoted to examples of how to use these commands.

SETTING ALL PIXELS IN A RECTANGLE TO THE SAME COLOR-INDEX

When the 4113 is first turned on, all its bit planes are assigned to Surface 1; thus Surface 1 can have color-indices in the range from 0 to 7. (If the terminal has four bit planes—Option 21—then color-indices can range from 0 to 15.) You can set all the pixels of Surface 1 to index four with the following commands:

<begin-pixel-operations : 1, 11, 3>
<rectangle-fill : (0,0), (639,479), 4>

The <begin-pixel-operations> command's three <int> parameters are 1, 11, and 3. The first parameter specifies Surface 1. The second parameter specifies "ALU mode 11." This means that pixels will be set to exactly the color-indices specified in subsequent <raster-write>, <runlength-write>, or <rectangle-fill> commands. (For more information about this parameter, see the command description in the 4110 Series Command Reference Manual.) The third parameter specifies "three bits per pixel." (You can ignore this parameter for now; it does not affect the <rectangle-fill> command.)

The <rectangle-fill> command also has three parameters. Two <xy> parameters specify the opposite corners of a rectangle on the current pixel writing surface (Surface 1, specified by the <begin-pixel-operations> command). The third parameter is an <int> specifying the color-index with which the terminal is to write to all the pixels in that rectangle.

To specify a rectangle filling all of raster memory space, the opposite corners should be (0,0) and (639,479). Note that the upper right corner is *not* (4095,3071); in all the pixel operations commands, coordinates are given in 640-by-480 raster memory space, *not* 4096-by-4096 terminal space.

Figure 7-28 shows the effect of several <rectangle-fill> commands. The first <rectangle-fill> command fills all of Surface 1, using color-index 4. The second command fills a square, using color-index 5. The third command fills another rectangle, using color-index 7.

INVERTED VIDEO

One way to achieve "inverted video" is to issue a <rectangle-fill> command while the terminal's "ALU mode" is set to mode 1. (The ALU mode is described next in this section.) In ALU mode 1, each bit of a pixel's color-index is inverted, changing ones to zeroes and zeroes to ones. Figure 7-29 shows how this is done.

ALU MODE

In Figure 7-29, the <begin-pixel-operations> command's second parameter specifies the ALU (arithmetic logic unit) writing mode. When a <rectangle-fill> (or <raster-write> or <runlength-write>) command is executed, the ALU mode determines how each pixel's color-index is modified.

Let A represent the pixel's old color-index; let B represent the color-index specified in the <rectangle-fill>, <raster-write>, or <runlength-write> command; and let C represent the pixel's new color-index after the command has been executed. Then C is some function of A and B.

In ALU mode 11 (used in Figure 7-28), C is set equal to B; thus the pixel is set to exactly the color-index specified in the <rectangle-fill>, <raster-write>, or <runlength-write> command.

In ALU mode 1, C—when expressed as a binary numeral—is the bit-by-bit complement of the binary numeral for A. In this mode (used in Figure 7-29), the pixel's new color-index does not depend on the color-index specified in the <rectangle-fill>, <raster-write>, or <runlength-write> command; it depends only on A, the pixel's previous color-index. It is this property of ALU mode 1 that lets you achieve an inverted video effect with the <rectangle-fill> command.

For more details on ALU mode, see the description in the 4110 Series Command Reference Manual of the <begin-pixel-operations> command.

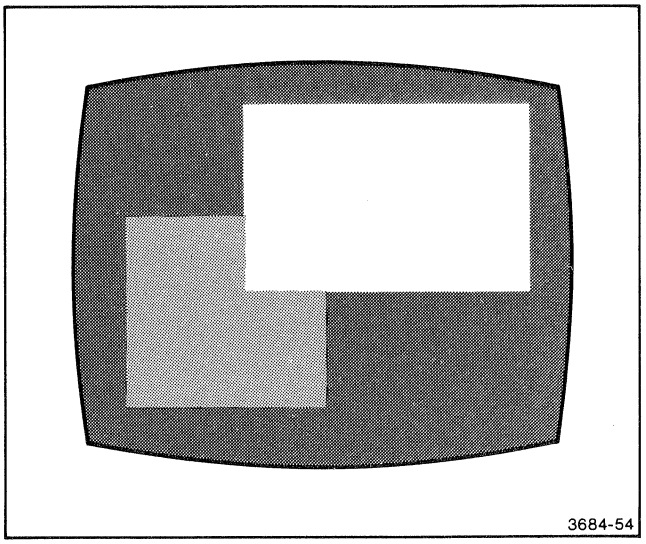


Figure 7-28. Effect of <Rectangle-Fill> Commands.

```
<begin—segment: 1>
<enter—vector—mode>
  <xy>
  <xy>
  .
  .
  .
<end—segment>
<begin—pixel—operations: 1, 1, 3>
<rectangle—fill: (50,50), (400,300), 4>
```

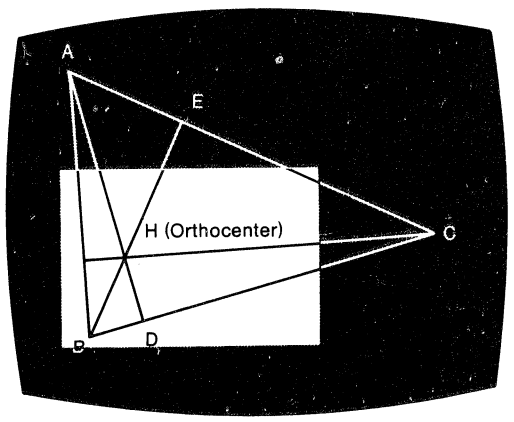


Figure 7-29. Inverted Video With the <Rectangle-Fill> Command.

WRITING INTO THE PIXEL VIEWPORT

Figure 7-30 shows commands that define a (very small) pixel viewport and write some color-indices into that pixel viewport.

Let's consider these commands, one by one.

<Begin-Pixel-Operations: 1, 11, 6>. The pixel viewport is to be on Surface 1. Pixels will be written using ALU mode 11 (set mode). The bits-per-pixel parameter for subsequent <raster-write> and <run-length-write> commands is 6.

<Set-Pixel-Viewport>. The lower left and upper right corners of the pixel viewport are at (100,100) and (109,109), respectively. (These coordinates are in 640-by-480 raster memory space.)

<Rectangle-Fill: (100,100), (109,109), 0>. The <rectangle-fill> command clears the pixel viewport, setting all pixels in that region to color-index zero. The coordinates in this command are in 640-by-480 raster memory space.

<Set-Pixel-Beam-Position: (3,4)>. Move the pixel beam to the point x= 3, y= 4. (These coordinates are relative to the lower left corner of the pixel viewport.)

<Raster-Write: 4, "4002">. Load four color-indices into four successive pixels. At the end of this command, the pixel beam position is at x= 7, y= 4. (These coordinates are relative to the lower left corner of the pixel viewport.)

<Set-Pixel-Beam-Position: (3,3)>. Move the pixel beam to the point (3,3). (These coordinates are relative to the lower left corner of the pixel viewport.)

<Raster-Write: 4, "4217">. Write the color-indices 4, 2, 1, 7 into four successive pixels. At the end of this operation, the pixel beam position is at (34,30)—relative to the lower left corner of the pixel viewport.

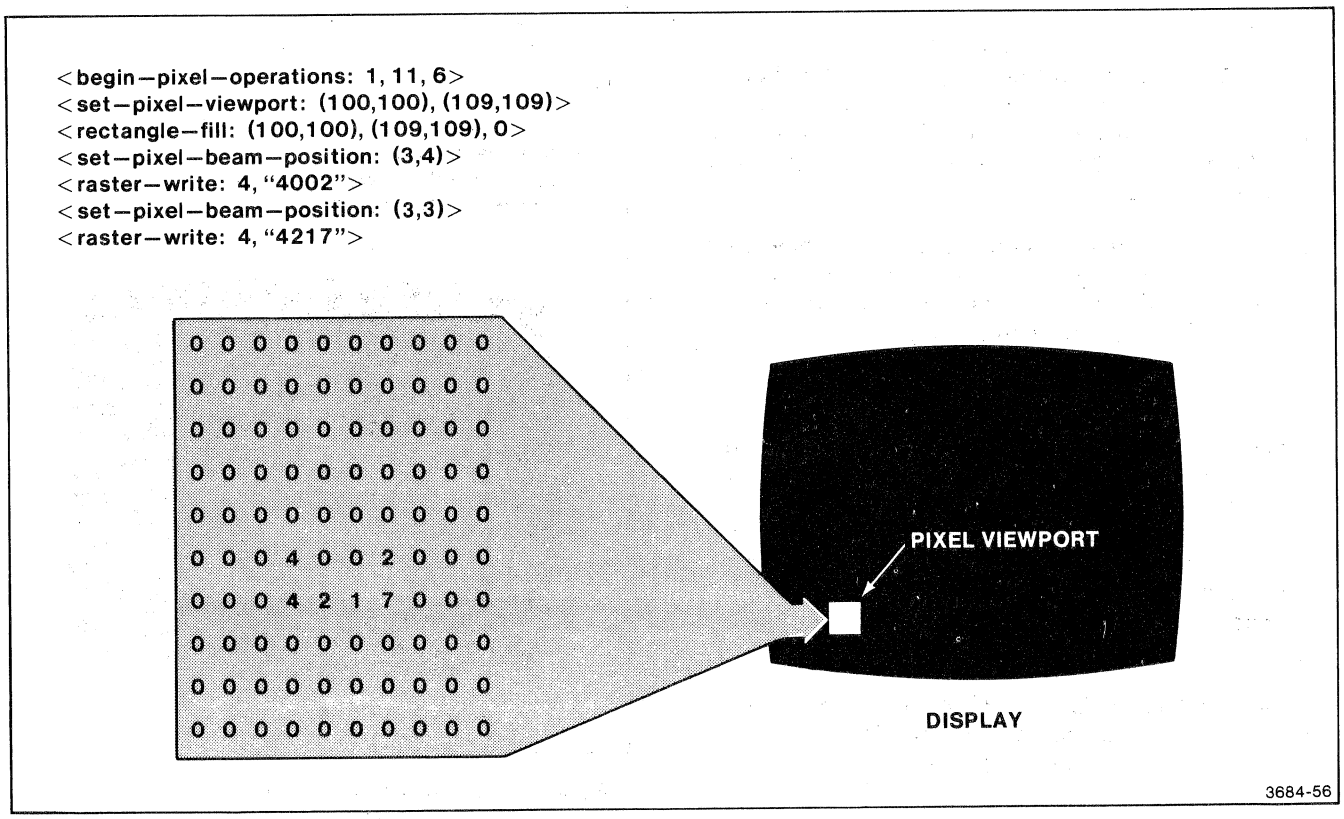


Figure 7-30. Writing Into the Pixel Viewport.

You can use <runlength-write> commands as well as <raster-write> commands in the pixel viewport. Figure 7-31 shows how.

As in the previous figure, Figure 7-31 uses <begin-pixel-operations> and <set-pixel-viewport> commands to define a pixel viewport on Surface 1 that is ten pixels wide and ten pixels high. As before, a <rectangle-fill> command is used to set all pixels in the pixel viewport to color-index zero. This time, however, there is no <set-pixel-beam-position> command, so the pixel beam position starts at the upper left corner of the pixel viewport. Also, this time the bits-per-pixel parameter is set to 3.

The <runlength-write> command has four run codes in its <int-array> parameter. The first code, 160, calls for 20 pixels of color-index zero (20 x 2^3 + 0 = 160). The second code, 243, calls for 30 pixels of color-index three (30 x 2^3 + 3 = 243). The third code, 160, calls for 20 pixels of color-index zero; it is the same as the first code. The fourth code, 246, calls for 30 pixels of color-index 6 (30 x 2^3 + 6 = 246).

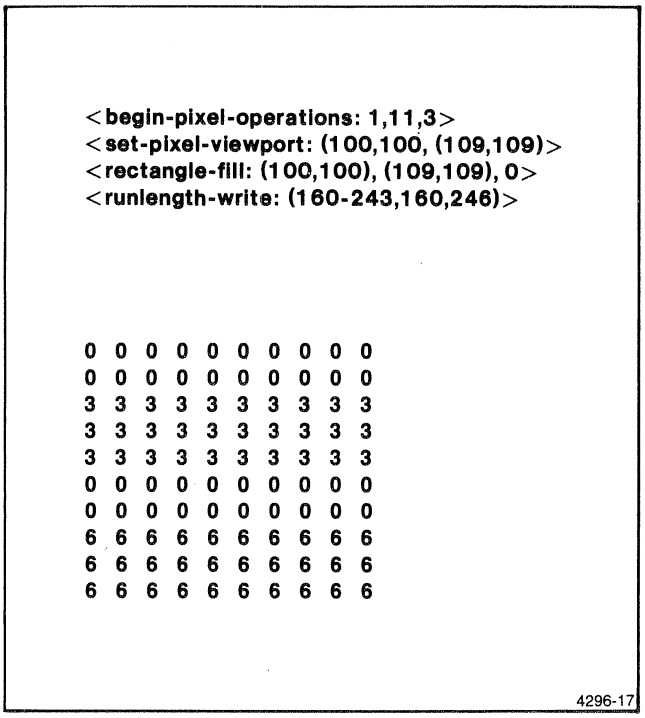


Figure 7-31. Using a <Runlength-Write> Command in the Pixel Viewport.

Section 8

GRAPHIC INPUT

INTRODUCTION

This section shows examples of GIN (graphic input) operations. Like all sections of this manual, it should be read with the 4110 Series Command Reference Manual close at hand. You will need the Command Reference Manual for detailed descriptions of the individual GIN commands.

In this section under the heading "Enabling for Graphic Input" is a description of the <enable-GIN> command and of GIN functions and GIN devices. Here also is a brief list of other GIN commands.

The next three major headings ("Locator Function," "Pick Function," and "Stroke Function") give examples of how to perform graphic input. These examples also show how to use other GIN features.

- **Locator function.** The example of the locator function also shows how to use the "gridding" and "rubberbanding" features.
- **Pick function.** The example of the pick function also shows how to insert <pick-ID>s in the display list for a segment.
- **Stroke function.** The example of the stroke function shows how to use "stroke filtering" and "inking" features.

By the time you have studied these examples, you should know how to do graphic input with the 4113 terminal. The remaining major head describes a more advanced topic: "Using Several GIN Devices at Once."

ENABLING FOR GRAPHIC INPUT

<ENABLE-GIN> COMMAND

The <enable-GIN> command enables the terminal for graphic input. The command has this syntax:

<enable-GIN> = (ESC)(I)(E)<int><int+>

Device-Function Code

The first parameter is an <int> holding a *device-function code*. This code specifies the *GIN device* which is enabled, and the *GIN function* for which that device is enabled.

Table 8-1 lists the valid device-function codes.

Table 8-1

DEVICE-FUNCTION ID CODE NUMBERS

Code	Device-Function Code
0	Thumbwheels-Locator
1	Thumbwheels-Pick
8	Tablet-Locator
9	Tablet-Pick
10	Tablet-Stroke
24	Plotter at Port 0 — Locator
25	Plotter at Port 0 — Pick
32	Plotter at Port 1 — Locator
33	Plotter at Port 1 — Pick
40	Plotter at Port 2 — Locator
41	Plotter at Port 2 — Pick

Number of GIN Events

The <enable-GIN> command's second parameter specifies for how many *GIN events* the device is being enabled. You can enable a device "permanently" (that is, until the terminal is turned off or a <disable-GIN> command is received) by specifying a large number, such as 65535, as the number of GIN events.

Each GIN event occurs when the terminal receives a single point's location from the operator and sends to the host a report including that position. If the graphic input device is the thumbwheels, the operator enters a point (signals a GIN event) by typing a character on the keyboard. For the optional graphics tablet, the operator signals a GIN event with the tablet pen or optional cursor. With the optional plotter, the operator signals a GIN event with a switch on the plotter. In any case, a GIN event occurs when a point's position is entered for graphic input.

The terminal remains enabled for graphic input from the specified device until one of the following occurs:

- The <enable-GIN> command's second parameter is satisfied. That is, the specified number of GIN events have occurred.
- A <disable-GIN> command is received.
- A <cancel> command is received, or the operator presses the CANCEL key.

Examples

<enable-GIN: 0, 1>	Enables the thumbweels device and locator function for one GIN event.
<enable-GIN: 1, 3>	Enables the thumbwheels for three pick events.
<enable-GIN: 8, 30>	Enables the tablet for 30 locator events.
<enable-GIN: 10, 65535>	Enables the tablet for a large number of stroke events.

GIN DEVICES

There are three types of GIN devices: *thumbwheels*, *tablet*, and *plotter*.

Thumbwheels

The operator uses the thumbwheels to position the graphic cursor and then signals a GIN event by pressing any ASCII key on the keyboard.

Tablet

The operator positions the graphic cursor by moving a stylus or tablet cursor over the surface of the optional graphic tablet. The operator signals a GIN event by pressing the stylus against the tablet, or by pressing a button on the tablet cursor. (In the case of the stroke function, GIN events occur automatically throughout the stroke. This is described later in this section.)

Plotter

If the 4113 is equipped with Option 10 (Three Port Peripheral Interface), then a TEKTRONIX 4662 or 4663 Interactive Digital Plotter may be attached to one of the three peripheral ports. Such a plotter may be specified as a graphic input device. If a plotter is the GIN device, then the operator positions the graphic cursor by moving the plotter pen with the plotter joystick. The operator signals a GIN event by pressing a switch on the plotter.

GIN FUNCTIONS

There are three GIN functions: *locator*, *pick*, and *stroke*.

Locator Function

When the locator function is enabled, a graphic cursor appears on the screen. (The default graphic cursor is a large pair of crosshairs; however, any segment can be used as the graphic cursor.)

The operator positions the cursor at some point on the screen, and then signals a "GIN event." Just how the operator does this depends on which GIN device has been enabled. For instance, if the thumbwheels are the GIN device, then the operator moves the cursor by manipulating the thumbwheels and signals a GIN event by pressing a keyboard key.

When the operator signals the GIN event, the terminal responds by sending a <GIN-locator-report> to the host computer. (An optional *signature character* is sent at the start of this report; this is described later in this section.) The <GIN-locator-report> tells the host (a) which key the operator pressed, and (b) the location of the graphic cursor in 4096-by-4096 terminal space.

Pick Function

When the pick function is enabled, a graphic cursor appears. The operator "picks" a segment (or part of a segment), and the terminal sends a <GIN-pick-report> to the host computer.

To pick a segment, the operator moves the graphic cursor to the desired segment (or part of a segment) and signals a GIN event. The operator does this just as for the locator function. For instance, he or she may move the thumbwheels to position the graphic cursor and press a key to signal the GIN event.

In response to the GIN event, the terminal sends a <GIN-pick-report> to the host computer. This report tells which key the operator pressed, where the cursor was when the operator pressed that key, which segment was picked, and what part of that segment was picked. The <GIN-pick-report>, like the <GIN-locator-report>, may be preceded by an optional signature character.

Stroke Function

The stroke function is valid only for the tablet.

When the stroke function is enabled, the graphic cursor appears at a point on the screen which corresponds to the position on the tablet of the tablet pen or tablet cursor. If you want to rely only on the tablet pen or cursor and would rather not see a graphic cursor on the screen, then use an empty segment as the graphic cursor.

Each "stroke" is deemed to be many separate GIN events, each of which causes a <GIN-stroke-report> to be sent to the host computer. The operator begins a stroke by pressing the tablet pen against the tablet, or by pressing a button on the tablet cursor. The operator then moves the pen (or cursor) across the tablet surface and ends the stroke by releasing pressure on the pen (or cursor button).

During the stroke, the terminal sends many <GIN-stroke-report>s to the host computer. The format of these reports, and details of when they are sent, are described later in this section.

<DISABLE-GIN> COMMAND

The <enable-GIN> command specifies a number of GIN events for which the GIN device and function are enabled. You can, however, disable graphic input from that device-function combination before all those GIN events have occurred. To do so, use the <disable-GIN> command:

<disable-GIN> = (ESC)(I)(D)<int>

Here, the <int> parameter is the device-function code for the GIN device and function which are to be disabled.

OTHER GIN COMMANDS

There are a variety of other GIN commands, which set other parameters associated with graphic input operations. These let you select alternate graphic cursors, choose the signature characters used in GIN reports, constrain the cursor to lie only on the intersection points of an imaginary grid, enable or disable "rubberbanding" and "inking," and so on. These commands are demonstrated in the examples which follow.

These commands pertain to GIN operations:

- <enable-GIN>
- <disable-GIN>
- <set-GIN-cursor>
- <set-GIN-gridding>
- <set-GIN-rubberbanding>
- <set-GIN-stroke-filtering>
- <set-pick-ID>
- <set-report-sig-chars>
- <set-report-EOM-frequency>
- <set-report-max-line-length>

You can find detailed information on these commands in the 4110 Series Command Reference Manual.

LOCATOR FUNCTION

The following example of the locator function also shows how to use the terminal's "gridding" and "rubberbanding" features.

PREPARING FOR GRAPHIC INPUT

To prepare the terminal for graphic input from the thumbwheels, using the locator function, you might issue the following commands:

<set-report-EOL-string: (13)>
<set-report-sig-chars: 0, 87, 119>
<set-report-EOM-frequency: 1>
<set-GIN-gridding: 0, 100, 100>
<set-GIN-rubberbanding: 0, 1>
<enable-GIN: 0, 5>

The <set-report-EOL-string> command sets the terminal's end-of-line string to be the single (CR) character, which has a decimal equivalent of 13:

<set-report-EOL-string: (13)>
= (ESC)(N)(T)<int-array: (13)>
= (ESC)(N)(T)<int: 1><int: 13>
= (ESC)(N)(T)(1)(=)

The <set-report-sig-chars> sets the "signature characters" used in the <GIN-report-sequence> which the terminal sends to the host. The command sets signature characters for device-function code zero: thumbwheels device, locator function. The <sig-char> sent before each <GIN-locator-report> is (W), which has a decimal equivalent of 87. The <term-sig-char>, which marks the end of the <GIN-report-sequence>, is (w), which has a decimal equivalent of 119.

<set-report-sig-chars: 0, 87, 119>
= (ESC)(I)(S)<int: 0><int: 87><int: 119>
= (ESC)(I)(S)(0)(E)(7)(G)(7)

The <set-report-EOM-frequency> command causes the terminal to send an <EOM-indicator> after each <GIN-locator-report> in the <GIN-report-sequence>. If the terminal is not in block mode, this <EOM-indicator> is just the current <EOL-string>, which has been set to (CR).

<set-report-EOM-frequency: 1>
= (ESC)(I)(M)<int: 1>
= (ESC)(I)(M)(1)

The <set-GIN-gridding> command enables gridding for device-function code zero. The graphic cursor is constrained to points in terminal space which have x- and y-coordinates that are both multiples of 100.

<set-GIN-gridding: 0, 100, 100>
= (ESC)(I)(G)<int: 0><int: 100><int: 100>
= (ESC)(I)(G)(0)(F)(4)(F)(4)

The <set-GIN-rubberbanding> command enables the "rubberbanding" feature for device-function code zero. After the first point is entered, the terminal displays a "rubberband line" between the previous point and the current cursor position.

<set-GIN-rubberbanding: 0, 1>
= (ESC)(I)(R)<int: 0><int: 1>
= (ESC)(I)(R)(0)(1)

After all these GIN parameters have been set, the <enable-GIN> command enables device-function code zero (thumbwheels device, locator function) for five GIN events.

<enable-GIN: 0, 5>
= (ESC)(I)(E)<int: 0><int: 5>
= (ESC)(I)(E)(0)(5)

OPERATOR AND HOST INTERACTION

The following steps show one way the preceding commands might be used in a host application program:

1. The program begins by issuing a <begin-segment> command and instructing the operator to "type M to move, D to draw, or X to exit this program."

2. The program then sets various graphic input parameters, and enables the device-locator function for a large number of GIN events. To do this, the program sends the following commands to the terminal:

   <set-report-sig-chars: 0, 87, 119>
   <set-report-EOM-frequency: 1>
   <set-GIN-gridding: 0, 1>
   <set-GIN-rubberbanding: 0, 1>
   <enable-GIN: 0, 32767>

3. As the terminal executes the <enable-GIN> command, it displays the crosshair cursor.

4. The operator moves the thumbwheels, using them to position the cursor at a desired point on the screen.

   Because gridding is enabled, the cursor only moves to points on an invisible grid. That is, it only moves to points which have x- and y-coordinates that are both multiples of 100.

   After the first point has been entered, the terminal displays a "rubberband line" from the last point entered to the current cursor location.

5. The operator enters each point by pressing a keyboard key. (The operator has been instructed to press M for a "move," D for a "draw," and X to exit the program.)

6. When the operator presses a key, the terminal blinks the crosshair cursor and sends a <GIN-locator-report> to the host computer. Each <GIN-locator-report> is preceded by (W) — the signature character — and followed by (CR) — the end-of-line string. Figure 8-1 shows a typical <GIN-report-sequence> for the locator GIN function.

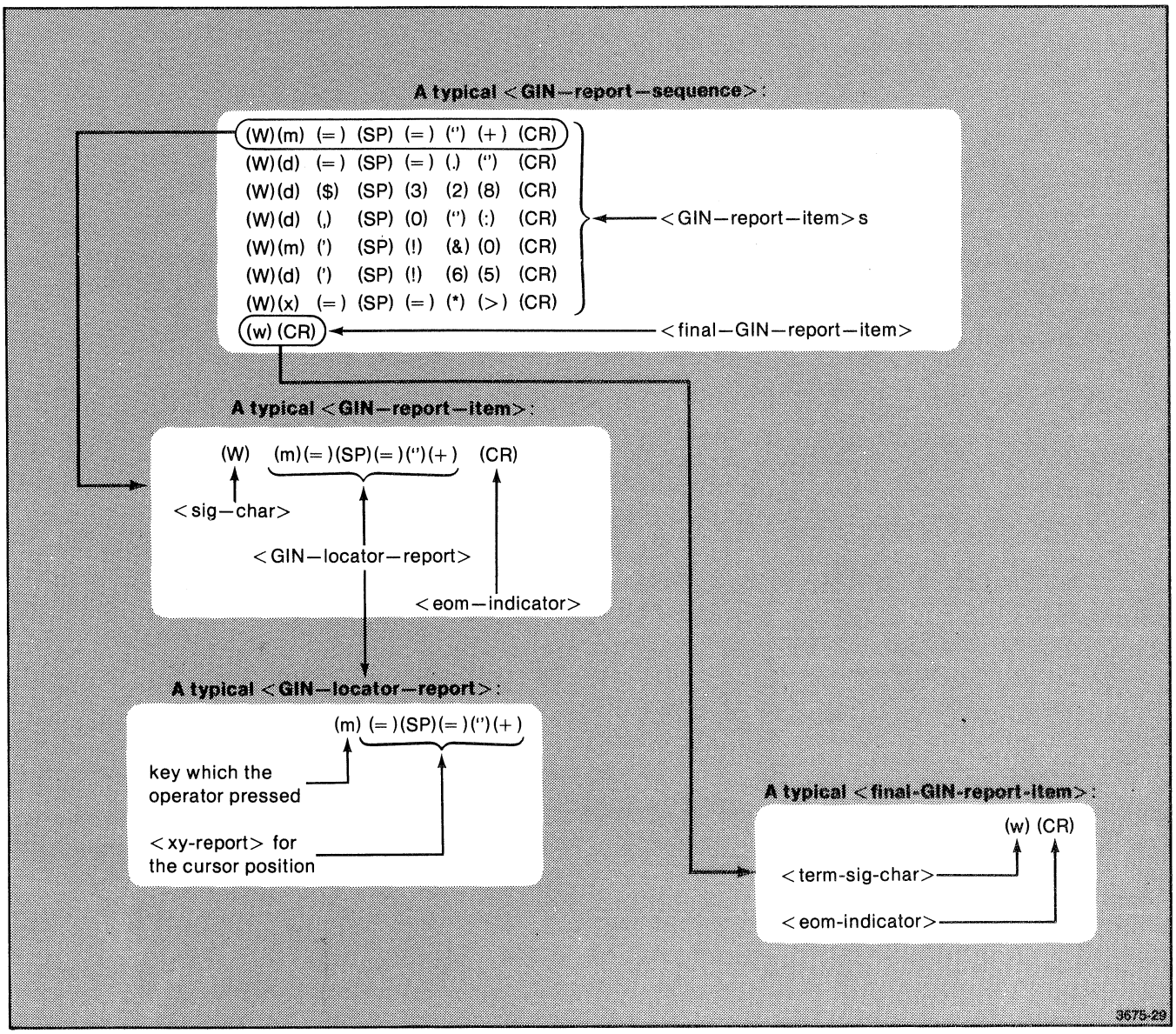


Figure 8-1. Typical <GIN-Report-Sequence> for the Locator Function.

7. The host applications program parses (reads) the <GIN-report-item> and takes appropriate action.

   To parse the <GIN-report-item>, the program reads characters coming from the terminal until it finds the signature character (W). It then calls a subroutine to parse the following six characters, which comprise a <GIN-locator-report>.

   To parse the <GIN-locator-report>, the host reads the first character, which tells which key the operator pressed. It then calls another subprogram to parse the next five characters, which comprise an <xy-report>.

   When done parsing the <GIN-locator-report>, the host knows (a) which key the operator pressed, and (b) where the cursor was when the operator pressed that key. Based on this information, the host then takes appropriate action.

   If a "move" was requested (operator typed M or m), the host issues a <move> command for the point just reported as the cursor position. (Alternatively, it could issue an <enter-vector-mode> command and an <xy> parameter.)

   If a "draw" was requested (operator typed D or d), the host issues a <draw> command for the point just reported as the cursor position.

   If an "exit" was requested (operator typed X or x), the host issues a <disable-GIN: 0> command. This disables graphic input for the thumbwheels device and locator function.

   If the operator typed some other character, then the host sends instructions to the operator.

8. Steps 4 through 7 are repeated again and again, until the operator signals an "exit" from the program. (That is, until the operator types X or x.)

9. When the operator types X or x, the terminal stops displaying the graphic cursor and sends the host a <final-GIN-report-item>. This <final-GIN-report-item> consists of the <term-sig-char> (the lowercase w character) and an <EOM-indicator> (the carriage return character, which is the current end-of-line string).

10. The host parses the <final-GIN-report-item> and uses it as a signal to exit its program loop.

11. The host sends an <end-segment> command to the terminal. All the <move>s and <draw>s which it sent during the graphic input operation are now included in a segment.

12. This is the end of this graphic input example. The example program exits, returning control to the host operating system.

Figure 8-2 shows a PASCAL program fragment for this graphic input example. (The complete PASCAL program is reproduced in Appendix C.)

```
.
.
.
PROCEDURE ParseGinReportItemAndTakeAppropriateAction; {}{}{}{}{}{}{}{}{}{}
{} BEGIN                                                                  {}
{} {** Look for signature character **}                                   {}
{}    WHILE NOT (TTY^ IN ['W', 'w' ]){* while next char is not W *}       {}
{}       DO BEGIN {** skip past the non-W character **}                   {}
{}          IF Eoln(TTY) {** If next character is a (CR) **}              {}
{}             THEN                                                      {}
{}                Readln(TTY) {** advance to 1st char of next line **}   {}
{}             ELSE                                                      {}
{}                Get(TTY) {** advance to next char **}                  {}
{}       END;                                                             {}
{} Read(TTY,SignatureChar);                                               {}
{} IF (SignatureChar = 'W')                                               {}
{}    THEN {** Parse a <GIN-locator-report>. **}                          {}
{}       BEGIN                                                            {}
{}       Read(TTY,KeyChar);                                               {}
{}       ParseAnXyReport(Xy);                                             {}
{}       IF KeyChar IN ['M','m','D','d','X','x']                          {}
{}          THEN {** take the appropriate action **}                      {}
{}             CASE KeyChar OF                                           {}
{}             {} 'M', 'm' : Move(Xy);                                   {}
{}             {} 'D', 'd' : Draw(Xy);                                   {}
{}             {} 'X', 'x' : DisableGin(0);                              {}
{}             {} END {** of CASE statement **}                          {}
{}          ELSE  {** If he typed an invalid key char **}                {}
{}             DisplayInstructions;                                      {}
{}       END;                                                             {}
{} END; {* of ParseGinReportItemAndTakeAppropriateAction procedure *} {}
{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}

BEGIN {** Main program **}
{** Prepare dialog area **}
   EnableDialogArea(1);
   SetDialogAreaVisibility(1);
{** Prepare for GIN **}
   {** <set-EOL-string : (13)> **}
      InitializeNewIntArray(IntArray);
      AppendToIntArray(IntArray,13);
      SetEolString(IntArray);
   SetReportSigChars(0,87,119); {* (W), (w) *}
   SetReportFomFrequency(1);    {* "more frequent " *}
   SetReportMaxLineLength(72);
   SetGinGridding(0,100,100);   {* enable gridding **}
   SetGinRubberbanding(0,1);    {* enable rubberbanding *}
{** Tell operator what to do **}
   Handshake; {* Be sure previous commands have been executed *}
   DisplayInstructions;
{** Open a segment **}
   DeleteSegment(-1); {** Delete all old segments. **}
   BeginSegment(1);
{** Enable for GIN **}
   EnableGin(0,32767); {** "32767" means "many points" **}
{** Loop until a <term-sig-char> is received. **}
   REPEAT
      ParseGinReportItemAndTakeAppropriateAction
   UNTIL SignatureChar = 'w';
{** Close the segment. **}
   EndSegment;

END.
```

3675-30

Figure 8-2. Graphic Input Example: Thumbwheels-Locator Device-Function Code.

PICK FUNCTION

INTRODUCTION

The graphic input "pick" function lets the operator choose one segment from among several which may be defined. For instance, in a drafting application, the operator might pick one of several circuit symbols.

A typical pick operation proceeds as follows:

1. The host computer enables a graphic input device for the pick function.
2. Using the enabled device (thumbwheels, tablet, or plotter), the operator moves the graphic cursor until it is at the image of the segment which is to be picked. The operator then presses a keyboard key (or plotter switch, or tablet pen, etc.) to signal the "pick event."
3. The terminal responds to the pick event by sending a <GIN-pick-report> to the host computer. This report tells the host (a) which key the operator pressed, (b) which segment the operator selected, (c) which *part* of that segment was selected, and (d) the graphic cursor location.

PREPARING SEGMENTS FOR PICKING

For a segment to be picked, several conditions must be met:

- Obviously, the segment must exist: it must have been defined with <begin-segment> and <end-segment> commands.
- The segment must be *detectable* — that is, pickable. A segment's detectability is set by the <set-segment-detectability> command.
- Part of the segment must fall within the current *pick aperture*. The pick aperture is a rectangle whose center is at the current graphic cursor position. (The size of this rectangle is determined by the <set-pick-aperture> command.)
- The part of the segment which falls within the pick aperture must have a *pick ID number* (pick identification number) which is greater than zero. Parts of a segment may be given different pick ID numbers with <set-pick-ID> commands. These commands must be included in the segment definition at the time the segment is defined.

Figure 8-3 shows how to prepare segments for picking.

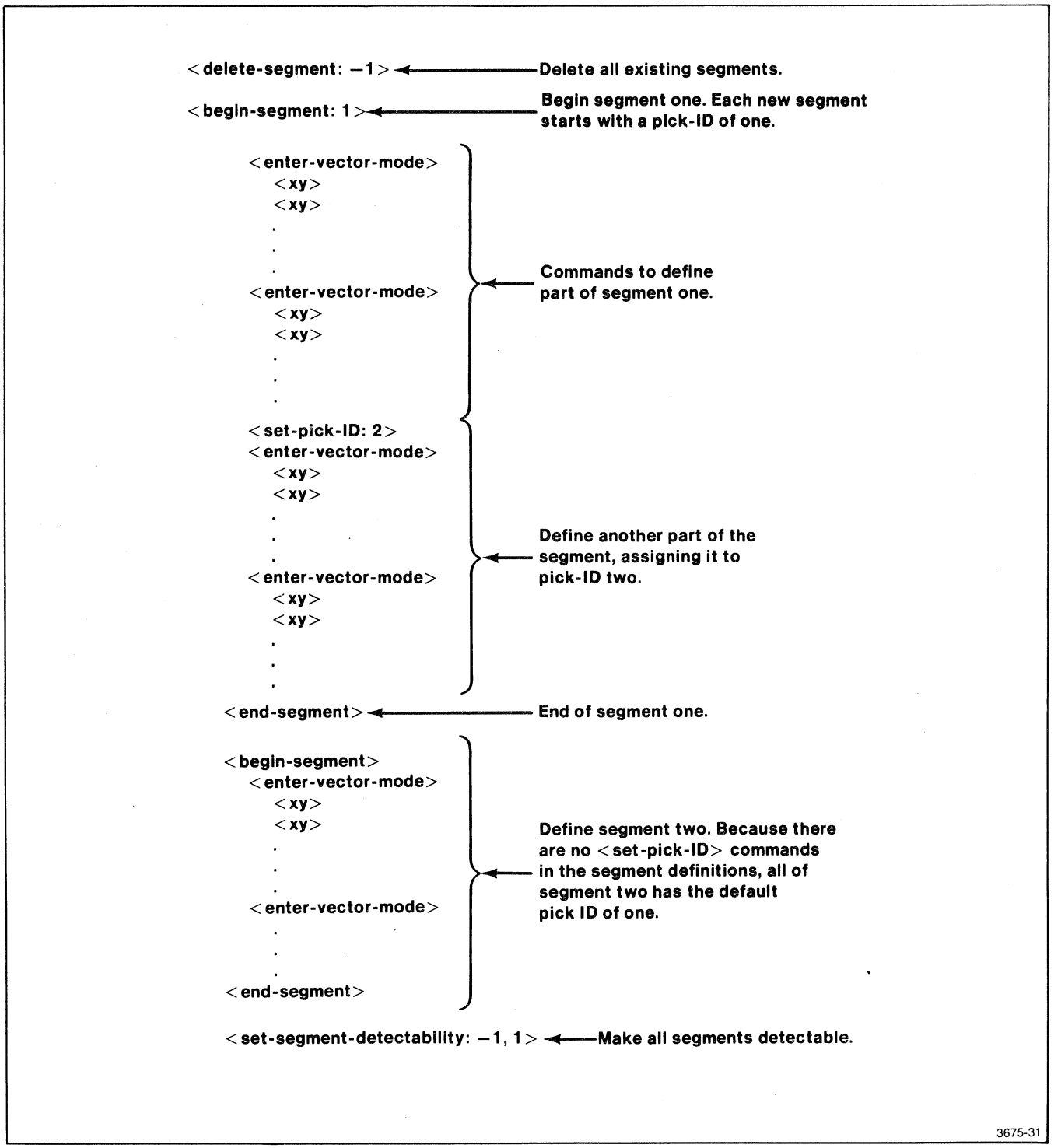


Figure 8-3. Preparing Segments for Picking.

OPERATOR AND HOST INTERACTION

The following steps show one way a host applications program might use the "pick" graphic input function:

1. The program begins by defining several segments at different locations on the screen.

2. Next, the program issues commands to prepare the terminal for five GIN pick events, using the tablet as the graphic input device ("tablet-pick" is device-function code 9):

   <set-EOL-string: (13)>
   <set-report-sig-chars: 9, 84, 116>
   <set-report-EOM-frequency: 1>
   <enable-GIN: 9, 5>

3. As the terminal executes the <enable-GIN> command, it displays the crosshair cursor.

4. The operator moves the tablet pen (or tablet cursor); this causes the crosshair cursor to move around the screen. When the cursor is at the image of the desired segment, the operator presses the tablet pen against the tablet (or presses a button on the tablet cursor). This signals a GIN pick event.

5. The terminal responds by sending a <GIN-report-item> to the host computer. The <GIN-report-item> consists of an uppercase letter T (the <sig-char>), a <GIN-pick-report>, and the carriage return character (the <EOM-indicator>).

   The <GIN-pick-report> tells the host (a) which segment the operator picked, (b) the pick ID number for the part of the segment which is near the graphic cursor, and (c) the location of the graphic cursor.

   The terminal also "blinks" the graphic cursor. This provides feedback to reassure the operator that the pick operation was successful.

6. The host program reads the <GIN-pick-report> and takes whatever action is appropriate.

7. In Step 2, the <enable-GIN> command specified five pick events. Therefore Steps 4, 5, and 6 are repeated four more times.

8. After sending the fifth <GIN-pick-report>, the terminal sends the <final-GIN-report-item>. This consists of the lowercase t character (the <term-sig-char>) and a carriage return character (the <EOM-indicator>).

Figure 8-4 shows a typical sequence of reports which the terminal might send the host for this graphic input example.

STROKE FUNCTION

INTRODUCTION

The graphic input "stroke" function is valid only when the Option 13 or Option 14 graphics tablet is the graphic input device.

The operator begins each stroke by pressing the pen against the tablet, or by placing an optional cursor on the tablet surface and pressing a switch on that cursor The operator then moves the pen (or cursor) along the tablet surface. As he does so, the terminal sends a stream of <GIN-stroke-report>s to the host computer. Each <GIN-stroke-report> reports one xy-coordinate to the host. Each such report counts as a separate GIN event for purposes of the <enable-GIN> command's "count" parameter.

A stroke ends when either of the following occurs:

- The <enable-GIN> command's count parameter is satisfied.
- The operator stops pressing the pen against the tablet, stops pressing the button on the tablet cursor, lifts the pen or cursor away from the tablet, or moves the pen or cursor outside the tablet's "presence area."

The stroke function is enabled by specifying device-function code 10 (tablet device, stroke function) in an <enable-GIN> command.

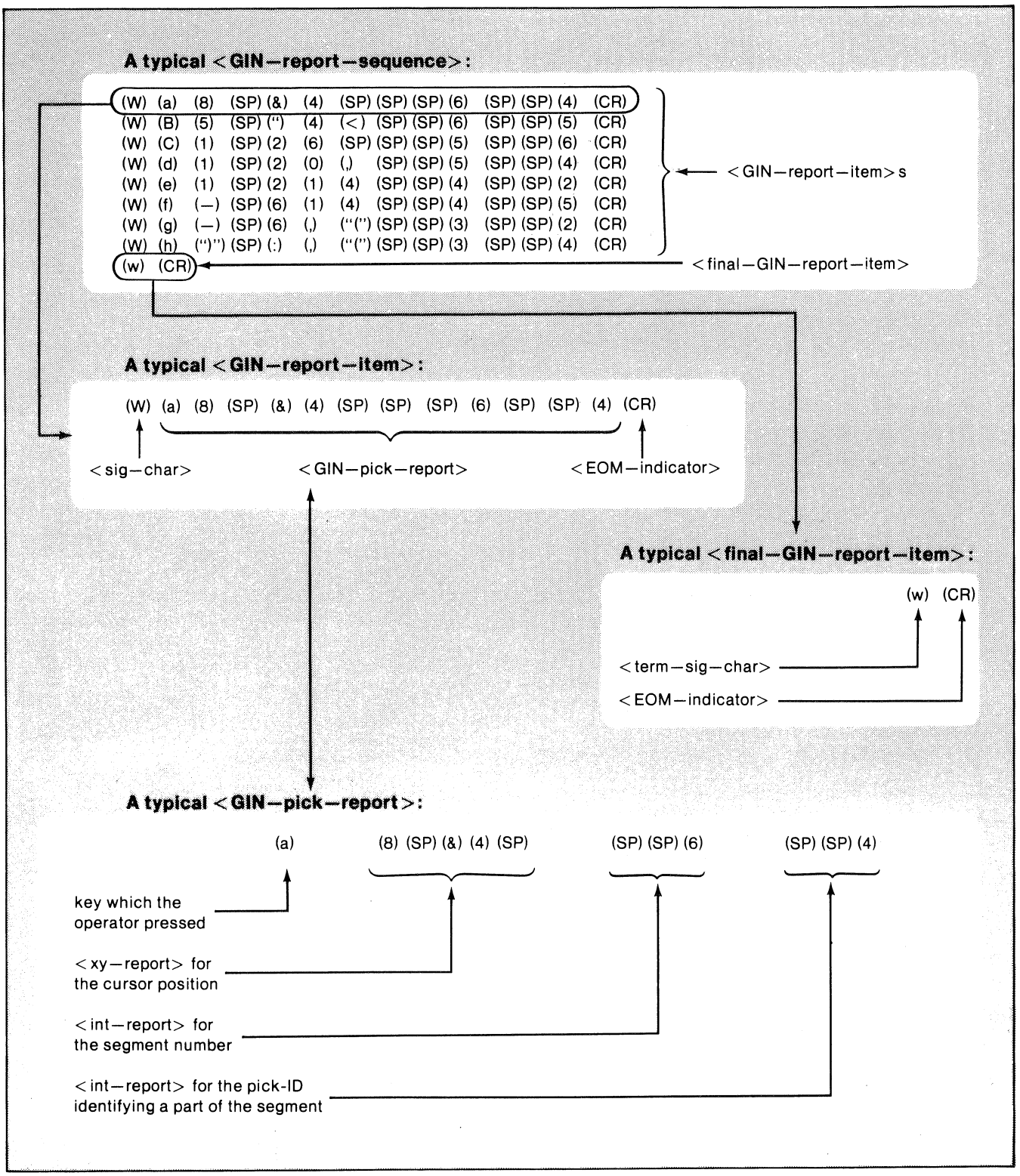


Figure 8-4. Typical <GIN-Report-Sequence> for the Pick Function.

STROKE FILTERING

One feature useful with strokes is "stroke filtering." Both "distance filtering" and "time filtering" are available.

The distance filter prevents a GIN event from occurring until the operator has moved the pen (or tablet cursor) a minimum distance. (For this purpose, "minimum distance" means "a minimum change in x or y.") With distance filtering, a stationary pen does not cause the terminal to send many <GIN-stroke-report>s for the same point on the tablet.

The time filter prevents a GIN event from occurring until a minimum time has elapsed since the last GIN event.

Stroke filtering is enabled with the <set-GIN-stroke-filtering> command:

<set-GIN-stroke-filtering>
= (ESC)(I)(F)
  <int: *device-function-code*>
  <int: *minimum-distance*>
  <int: *minimum-time*>

Here, the first <int> parameter should be 10, which is the device-function code for the tablet device and stroke function.

The second parameter specifies the distance filter. A new GIN event does not occur until the cursor has moved so that either its x-coordinate or its y-coordinate has changed by at least the distance specified in this parameter. If this parameter is zero, the distance filter is disabled.

The third parameter specifies the time filter, in milliseconds. A new GIN event does not occur until at least this number of milliseconds has elapsed since the preceding GIN event.

INKING

Another useful feature with strokes is "inking." For each stroke, the terminal draws line segments on its screen connecting the points specified in the <GIN-stroke-report>s for that stroke.

(The inking feature may also be used with the locator function. For details, see the descripion in the 4110 Series Command Reference Manual of the <set-GIN-inking> command.)

The inking feature is enabled with the <set-GIN-inking> command:

<set-GIN-inking> = (ESC)(I)(I)
  <int: *device-function-code*>
  <int: *inking-mode*>

The second parameter (inking mode) is one to enable inking and zero to disable inking.

STROKE REPORT FORMAT

During a stroke function, the terminal reports the coordinates of the tablet pen or cursor using the <GIN-report-sequence> format, in which the individual <GIN-stroke-report>s each report one coordinate position. The formal syntax is described in Section 4 of the 4110 Series Command Reference Manual, under "<GIN-Report-Sequence> Message Type" and "<GIN-Stroke-Report>."

A Typical Stroke Report Sequence

Figure 8-5 shows a typical stroke report sequence. This example assumes that the report-EOM-frequency setting is "more frequent," that the terminal is not in block mode, and that the <EOL-string> is set to (CR). Under these circumstances, each <GIN-Stroke-Report> is followed by a (CR). (For more information on the report-EOM-frequency setting, see Section 9. For information on the <EOL-string>, see Section 10.)

Signature Characters

In Figure 8-5, the <sig-char> and <term-sig-char> are (T) and (t), respectively. Thus, each <GIN-report-item> begins with the character (T), while the end of the <GIN-report-sequence> is marked with the character (t).

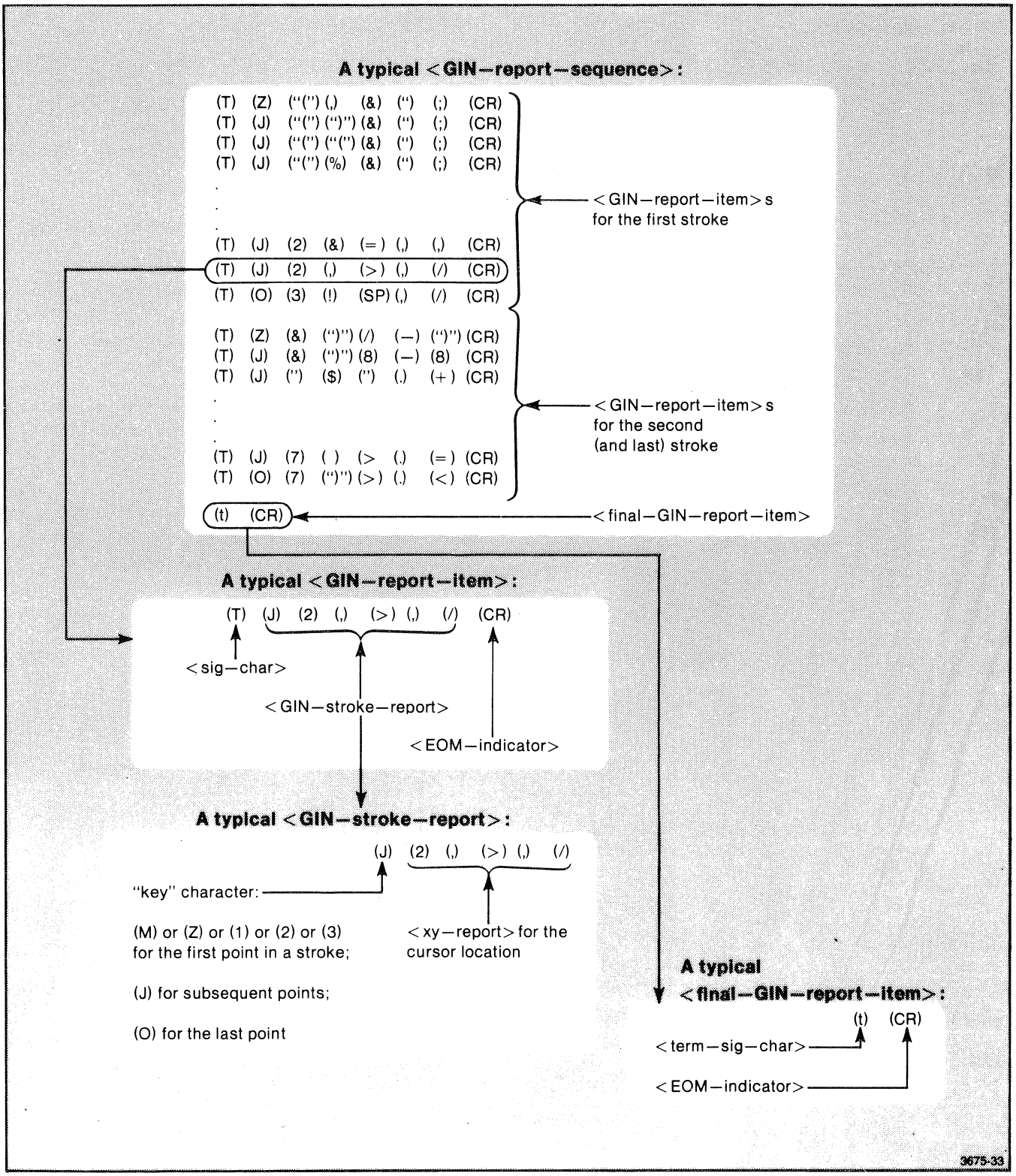


Figure 8-5. Typical <GIN-Report-Sequence> for the Stroke Function.

Key Characters

After the (T), each <GIN-stroke-report> begins with a "key" character. For the first point in a stroke, this is (M); for the last point in the stroke, it is (O). For intermediate points, the key character is (J).

If the operator used the tablet cursor rather than the tablet pen, then the key character for the first point in the stroke would be (Z), (1), (2), or (3), depending on which of the cursor buttons is pressed. The intermediate points would still have (J) for their key character, and the last point would still have (O) for its key character.

There is a command, <set-tablet-header-chars>, which can change the key characters for the last point and intermediate points of the stroke. This command does not change the key character for the first point of a stroke. See the 4110 Series Command Reference Manual for details.

<XY-Report>s

After the key character, each <GIN-stroke-report> contains five more characters. These comprise an <xy-report> telling the position of one point in the stroke. For more information on <xy-report>s, see the <xy-report> description in the 4110 Series Command Reference Manual.

Fitting More Than One Stroke Report on Each Line

If you like, you can cause more than one <GIN-report-item> to occur on each "line of text" that the terminal sends to the host. To do this, you would issue a <set-report-EOM-frequency: 0> command to make the terminal send (CR) characters "less frequently." You would also issue a <set-report-max-line-length> command to specify the maximum length for each line of the report. (These commands are described in Section 9.)

If the maximum report line length is 25, and the report EOM frequency is "less frequent," then a typical stroke report sequence is like that in Figure 8-6.

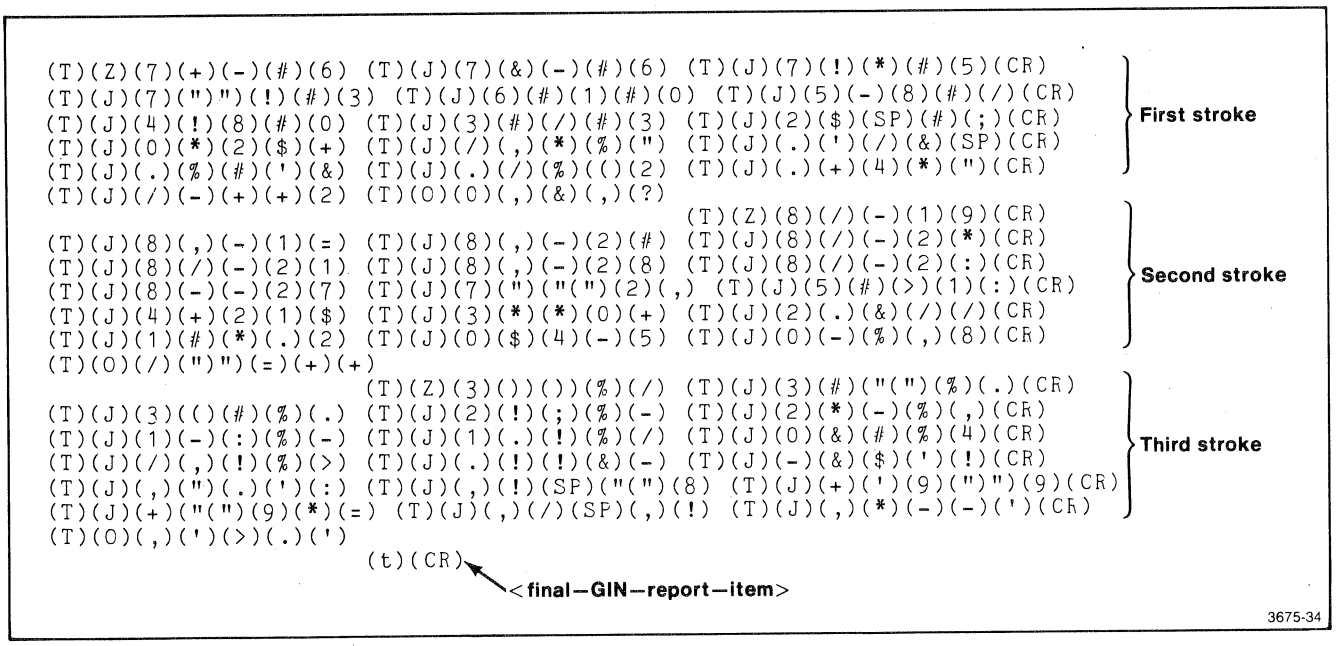


Figure 8-6. Fitting Multiple Stroke Reports on Each Line.

USING SEVERAL GIN DEVICES AT ONCE

You can enable several GIN devices at once. For instance, the terminal can simultaneously be enabled for graphic input from the thumbwheels device, locator function (device-function code 0) and the tablet device, pick function (device-function code 9).

SIGNATURE CHARACTERS

When several graphic input devices are enabled, the reports which they send to the host are interleaved. In order that the host may parse the reports correctly, each device-function code should have its own distinct signature characters. For instance, you might assign (W) and (w) as the <sig-char> and <term-sig-char> for the thumbwheels, and assign (T) and (t) as the corresponding signature characters for the tablet.

CURSORS

The different device-function codes should have different cursors. If you use the default crosshair cursor ("segment zero") for one device-function code, you should assign other segments to serve as the cursors for the other device-function codes. To do this, use the set-GIN-cursor> command:

<set-GIN-cursor>
= (ESC)(I)(C)<int><int>

Here, the first <int> parameter names the device-function code, and the second <int> parameter is the segment number for the segment being assigned as the graphics cursor.

Section 9

REQUESTING REPORTS FROM THE TERMINAL

This section describes the commands by which you can request reports from the terminal, and by which you can control the format of those reports. Not included here are descriptions of the graphic input commands; those are described in Section 8. Topics here are:

- **Controlling the format of reports.** This includes setting the signature characters, "EOM frequency," the maximum line length for reports, and so on. The commands described here affect GIN reports (described in Section 8) as well as the non-GIN reports described in this section.
- **Non-GIN reports.** This section also covers how to obtain information on the terminal's settings, on the status of devices attached to the terminal, and on segments stored in the terminal's RAM memory.

CONTROLLING THE FORMAT OF REPORTS

REPORT SYNTAX

For each report that the terminal sends the host computer, there is a syntax specified in the 4110 Series Command Reference Manual. You will need to consult that syntax description when you write a routine to parse that report. See the following descriptions in the Command Reference Manual:

- <GIN-report-sequence>
- <GIN-locator-report>
- <GIN-pick-report>
- <GIN-stroke-report>
- <Port-status-report>
- <Segment-status-report>
- <Terminal-settings-report>
- <4010-GIN-report>
- <4953-tablet-GIN-report>

<EOM-INDICATOR>S

Most of these reports have <EOM-indicator>s in their syntax. Some of the <EOM-indicator>s are optional parts of the reports; in the syntax definitions, these are enclosed in square brackets (e.g., "[<EOM-indicator>]"). Footnotes explain under what circumstances the optional <EOM-indicator>s are included in the report message.

If the terminal is in block mode (which requires Option 01), then it sends an <EOM-indicator> by terminating the block and setting the end-of-message bit in the <block-control-bytes>. This is described in Section 11.

If the terminal is not in block mode, then it replaces each <EOM-indicator> with its current <EOL-string> (end-of-line string). This is typically the single character, (CR), although other <EOL-string>s may be selected with the <set-EOL-string> command.

In either case (whether the terminal is in block mode or not), each <EOM-indicator> marks the end of a "line of text" in a report message which the terminal sends to the host.

<SET-REPORT-EOM-FREQUENCY> COMMAND

The <set-report-EOM-frequency> command lets you control how often the terminal sends <EOM-indicator>s in report messages.

Consider, for instance, the syntax specified for <GIN-report-sequence>. This syntax is specified in the 4110 Series Command Reference Manual. For this discussion, the pertinent part of the syntax is as follows:

<GIN-report-sequence>
= [<GIN-report-item> . . .]
  <final-GIN-report-item>

where

<GIN-report-item> = [<EOM-indicator>]
[<sig-char>]
<GIN-report>
[<EOM-indicator>]

By issuing a <set-report-EOM-frequency: 1> command, you can cause the terminal to send an <EOM-indicator> at the end of each <GIN-report-item>. (This is the <EOM-indicator> shown in **bold** type.) That way, if the terminal is not in block mode, each individual <GIN-report> is followed with a (CR) (or other <EOL-string>). If the terminal is in block mode, each individual <GIN-report> is in a block of its own.

By issuing a <set-report-EOM-frequency: 0> command, you can suppress the <EOM-indicator>s at the end of the <GIN-report-item>s. That way, several <GIN-report-item>s can fit on the same line of text.

The <set-report-EOM-frequency> command has this syntax:

<set-report-EOM-frequency> = (ESC)(I)(M)<int>

The <int> parameter is one if <EOM-indicator>s are to be sent "more frequently;" it is zero if <EOM-indicator>s are to be sent "less frequently."

<SET-REPORT-MAX-LINE-LENGTH> COMMAND

If you decide to allow several reports to be sent in the same line of text (using the <set-report-EOM-frequency:0> command), then the question arises, "How many reports should the terminal send on each line of text?" Another way to phrase this is, "What is the maximum length permitted for each line of text?"

You set the maximum line length permitted in reports with the <set-report-max-line-length> command:

<set-report-max-line-length>
= (ESC)(I)(L)<int>

Here, the <int> parameter specifies the maximum number of characters allowed on each line of the report message, not counting any characters in the <EOM-indicator> that terminates the line. (If the terminal is not in block mode, the <EOM-indicator> is the current <EOL-string>, as set by the <set-EOL-string> command.)

To see how the <set-report-max-line-length> command affects the report messages which the terminal sends to the host, consider once again, the <GIN-report-sequence> syntax:

<GIN-report-sequence> = [<GIN-report-item> . . .]
<final-GIN-report-item>

where

<GIN-report-item> = **[<EOM-indicator>]**
[<sig-char>]
<GIN-report>
[<EOM-indicator>]

If the <report-EOM-frequency> is set to "less frequent," then the terminal does not send the optional <EOM-indicator> at the end of each <GIN-report-item>. In that case, the <GIN-report-item> syntax is simplified:

<GIN-report-item> = **[<EOM-indicator>]**
[<sig-char>]
<GIN-report>

The optional <EOM-indicator> at the start of each <GIN-report-item> (shown above in **bold** type) is only sent if the terminal's "maximum report line length" is about to be exceeded. This maximum line length is determined by the <set-report-max-line-length> command.

Suppose, for instance, that the terminal is enabled for graphic input using the locator function, and that the maximum line length is set to 78 characters. Then each <GIN-report> in the <GIN-report-sequence> is a <GIN-locator-report>, and consists of six characters. If the <sig-char> is not (NUL), it is sent before each <GIN-report>, so each <GIN-report-item> has seven characters, not counting the optional <EOM-indicator>. Without exceeding the maximum line length, eleven seven-character <GIN-report-item>s can fit on each line.

When the terminal has sent eleven <GIN-report-item>s, it has filled up a line of text. If the terminal has a twelfth <GIN-report-item> to send, it begins by sending an <EOM-indicator>. This <EOM-indicator> serves to terminate the preceding line of text, so that the maximum line length is not exceeded. The remainder of the <GIN-report-item> — the <sig-char> and <GIN-locator-report> — occupies the first seven characters of the next line of text.

EXAMPLES

Figure 9-1 shows two typical <GIN-report-sequence>s for the locator function. In the first part of the figure, the EOM-frequency is "more frequent," so that each <GIN-report-item> is on a separate line of text. In the second part of the figure, the EOM-frequency is "less frequent," and the maximum report line length is 25; this fits three seven-character <GIN-report-item>s on each line of text.

```
(W)(M)(&)(SP)("(")(')(!)(CR)
(W)(D)(-)(SP)(")")(')(!)(CR)
(W)(D)(-)(SP)(")")(%)(+/)(CR)
(W)(D)(0)(SP)(-)(.)(")(CR)
(W)(D)(-)(SP)(")")(6)(5)(CR)
(W)(D)(-)(SP)(")")(5)(#)(CR)
(W)(D)(&)(SP)("(")(5)(#)(CR)
(W)(D)(&)(SP)("(")(')(!)(CR)
(W)(M)(&)(SP)("(")(2)(8)(CR)
(W)(D)(*)(SP)(%)(2)(8)(CR)
(W)(D)(*)(SP)(%)(1)(&)(CR)
(W)(D)(&)(SP)("(")(1)(&)(CR)
(W)(D)(SP)(SP)(SP)(0)(-)(CR)
(M)(W)(M)(SP)(SP)(SP)(3)(1)(CR)
(W)(D)(&)(SP)("(")(2)(8)(CR)
(W)(M)($)(SP)(6)(SP)(SP)(CR)
(W)(D)(')(SP)(!)(')(!)(CR)
(W)(M)(*)(SP)(+.)(5)(#)(CR)
(W)(D)(.)(SP)(:)(?)("(")(CR)
(W)(X)(#)(SP)($)(?)("(")(CR)
(W)(CR)
```

A. <Set—EOM—Frequency: 1>

```
(W)(M)(&)(SP)("(")(')(!)(W)(D)(-)(SP)(")")(')(!)(W)(D)(-)(SP)(")")(%)(+/)(CR)
(W)(D)(0)(SP)(-)(.)(")(W)(D)(-)(SP)(")")(6)(5)(W)(D)(-)(SP)(")")(5)(#)(CR)
(W)(D)(&)(SP)("(")(5)(#)(W)(D)(&)(SP)("(")(')(!)(W)(M)(&)(SP)("(")(2)(8)(CR)
(W)(D)(*)(SP)(%)(2)(8)(W)(D)(*)(SP)(%)(1)(&)(W)(D)(&)(SP)("(")(1)(&)(CR)
(W)(D)(SP)(SP)(SP)(0)(-)(M)(W)(M)(SP)(SP)(SP)(3)(1)(W)(D)(&)(SP)("(")(2)(8)(CR)
(W)(M)($)(SP)(6)(SP)(SP)(W)(D)(')(SP)(!)(')(!)(W)(M)(*)(SP)(+.)(5)(#)(CR)
(W)(D)(.)(SP)(:)(?)("(")(W)(X)(#)(SP)($)(?)("(")(W)(CR)
```

B. <Set—EOM—Frequency: 0>,<Set—Report—Max—Line—Length: 25>

3675-35A

Figure 9-1. Controlling the Format of a <GIN-Report-Sequence>.

Likewise, Figure 9-2 shows two <errors-report> sequences which the terminal might send in response to a <report-errors> command. (The <report-errors> command is described later in this section and in the Command Reference Manual. For details of the <errors-report> syntax, see the 4110 Series Command Reference Manual.)

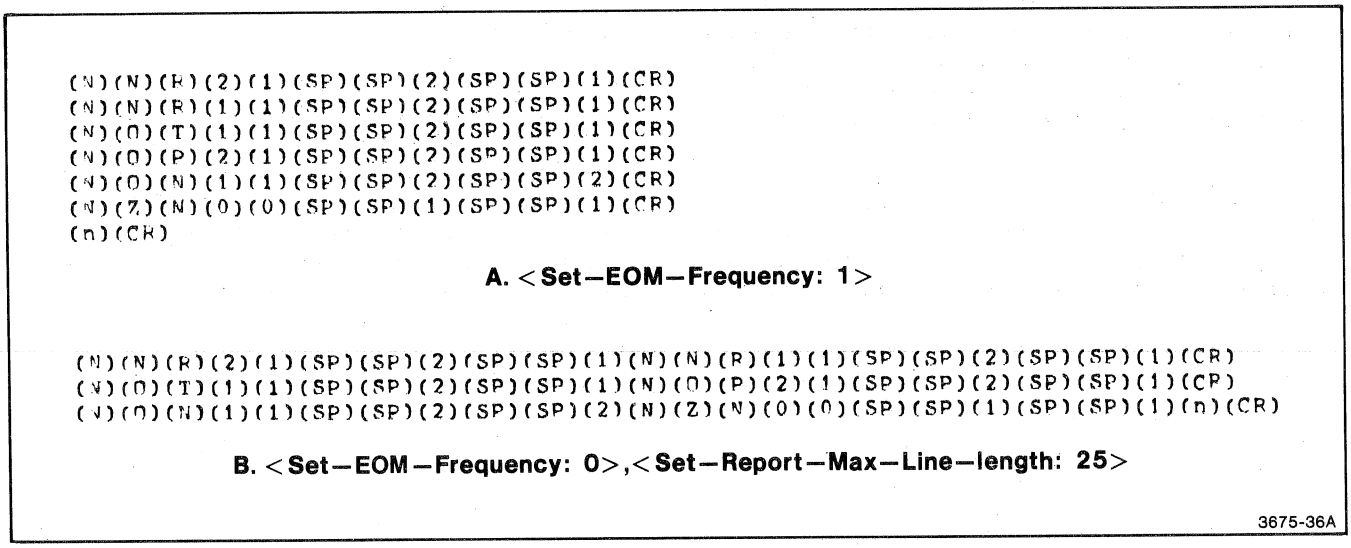


Figure 9-2. Controlling the Format of <Errors-Report> Messages.

<SET-REPORT-SIG-CHARS> COMMAND

The <set-report-sig-chars> command sets the <sig-char> and <term-sig-char> signature characters, which are used in <GIN-report-sequence> messages and also in non-GIN report messages. The command has the following syntax:

<set-report-sig-chars>
= (ESC)(I)(S)<int><int><int>

The first <int> parameter specifies the type of report for which the signature characters are being specified. For GIN reports, this is a GIN device-function code. (See Section 8 for information on device-function codes.) For other reports ("non-GIN" reports), this parameter is minus three. (All non-GIN reports have the same <sig-char> and <term-sig-char> signature characters.)

If this parameter is minus one, then the <set-report-sig-chars> command sets the signature characters for all reports: all GIN device-function codes, and all non-GIN reports as well.

The second and third <int> parameters specify the ASCII characters to be used as the <sig-char> and <term-sig-char> in the reports of the type specified by the first parameter. Each character is specified by its numeric equivalent. For instance, <int: 65> represents the (A) character, since the ASCII decimal equivalent of A is 65.

Setting a signature character to (NUL) — ASCII decimal equivalent of zero — causes that character not to be sent in reports to the host computer.

For more information, see the description in the 4110 Series Command Reference Manual of the <set-report-sig-chars> command.

NON-GIN REPORTS

Besides the commands for graphic input, the following commands cause the terminal to send report messages to the host computer:

- <report-device-status>
- <report-errors>
- <report-port-status>
- <report-segment-status>
- <report-terminal-settings>
- <report-4010-status>

The following descriptions summarize the purposes of these commands. For more information, see the descriptions of these commands in the 4110 Series Command Reference Manual.

<REPORT-DEVICE-STATUS> COMMAND

The <report-device-status> command causes the terminal to send a <device-status-report> to the host computer. The <device-status-report> tells the host certain status information about the peripheral device specified in the <report-device-status> command. The command has the following syntax:

<report-device-status> = (ESC)(J)(O)<string>

If Option 10 (Three Port Peripheral Interface) is installed, the <string> may be "P0:", "P1:", or "P2:", to specify one of the RS-232 peripheral ports. If Option 42 or 43 (disk drive or drives) is installed, the <string> may be "F0:" or "F1:", to specify a particular flexible disk drive.

For more details, see the following descriptions in the 4110 Series Command Reference Manual:

- <Report-device-status> command
- <Device-status-report> message type

<REPORT-ERRORS> COMMAND

The <report-errors> command causes the terminal to send an <errors-report> message to the host computer. In that message, the terminal reports the eight most-recently detected error codes, their severity levels, and how many times each error was detected. The command has the following syntax:

<report-errors> = (ESC)(K)(Q)

For details, see the following descriptions in the 4110 Series Command Reference Manual:

- <Report-errors> command
- <Errors-report> message type

See also Appendix C of the Command Reference Manual, "Error Codes."

<REPORT-PORT-STATUS> COMMAND

The <report-port-status> command is available only if Option 10, the Three Port Peripheral Interface, is installed. The command has the following syntax:

<report-port-status> = (ESC)(P)(Q)<string>

Here, the <string> parameter is "P0:", "P1:", or "P2:". It names the RS-232 peripheral port for which a status report is requested. In response to the command, the terminal sends a <port-status-report> message to the host computer. This message contains information about various settings for the specified peripheral port: baud rate, parity, flagging mode, etc.

For details, see the following descriptions in the 4110 Series Command Reference Manual:

- <Report-port-status> command
- <Port-status-report> message type

<REPORT-SEGMENT-STATUS> COMMAND

The <report-segment-status> command causes the terminal to send a <segment-status-report> message to the host computer. The command has the following syntax:

<report-segment-status>
= (ESC)(S)(Q)<int><char-array>

Here, the <int> parameter names the segment for which status information is requested. The parameter may be in the range from 1 to 32767, in which case information about one specified segment is returned. Or, the parameter may be one of the "special segment numbers" described in Section 6: "−1" means "all segments," "−2" means "default values for segments not yet defined," and "−3" means "all segments in the current segment matching class."

The <char-array> tells the terminal which information about the specified segment (or segments) is desired.

For details, see the following descriptions in the 4110 Series Command Reference Manual:

- <Report-segment-status> command
- <Segment-status-report> message type

<REPORT-TERMINAL-SETTINGS> COMMAND

The <report-terminal-settings> command is an extremely versatile command by which the host can obtain a wealth of information about the terminal's settings. The command has the following syntax:

<report-terminal-settings>
= (ESC)(I)(Q)<char><char>

Here, the two <char> parameters comprise the op code for one of the terminal's commands. The terminal responds by sending the host a <terminal-settings-report> telling what the current values are for the specified command's parameters.

Example: Querying the Terminal for Its Baud Rate Settings

For instance, the <set-baud-rates> command has this syntax:

<set-baud-rates>
= (ESC)(N)(R)<int+><int+>

Since the op code for this command is NR, the host can learn the terminal's baud rates by issuing the following command:

<report-terminal-settings: (N),(R)>
= (ESC)(I)(Q)(N)(R)

In response to this command, the terminal sends the host a <terminal-settings-report>, as follows:

<terminal-settings-report>
= [<sig-char>]
(N)(R)
<int-report: *transmit-rate*>
<int-report: *receive-rate*>
<EOM-indicator>

Assume that the <sig-char> for non-GIN reports is (T), that the terminal's <EOL-string> is (CR), that the terminal is not in block mode, and that it is set to transmit characters at 300 baud and receive them at 600 baud. Under these circumstances, the <terminal-settings-report> would be as follows:

<terminal-settings-report>
= (T)
(N)(R)
<int-report: 300>
<int-report: 600>
(CR)

= (T)
(N)(R)
(SP)(1)(>)
(SP)(C)(<)
(CR)

= (T)(N)(R)(SP)(1)(>)(SP)(C)(<)(CR)

Special Inquiry Codes

There are also a number of "special inquiry codes," which can be used instead of command op codes in the <report-terminal-settings> command. For instance, you can find out how much free memory the terminal has available by the "?M" inquiry code:

<report-terminal-settings: (?)(M)>
= (ESC)(I)(Q)(?)(M)

For information on these inquiry codes, and for full details on the <report-terminal-settings> command, see the following descriptions in the 4110 Series Command Reference Manual:

- <Report-terminal-settings> command
- <Terminal-settings-report> message type

<REPORT-4010-STATUS> COMMAND

The <report-4010-status> command is provided for compatibility with TEKTRONIX 4010 Series terminals. It has the following syntax:

<report-4010-status> = (ESC)(ENQ)

In response to this command, the terminal sends a <4010-status-report> to the host computer. For details, see the following descriptions in the 4110 Series Command Reference Manual:

- <Report-4010-status> command
- <4010-status-report> message type

Section 10

COMMUNICATIONS SETTINGS

INTRODUCTION

This section tells how to control the 4113's standard communications settings. Not included are the settings associated with Option 01 (Half Duplex and Block Mode); those settings are described in Section 11.

Topics discussed in this section are:

- The most important communications settings: data rates, echo, parity, and number of stop bits.
- Less important settings: the break time, and coping with (DEL) filler characters which some hosts send.
- Full duplex data communications.
- The terminal's communications input queue, and the handshaking protocols used to keep it from overflowing.
- The concept of "lines of text" in data sent from the terminal to the host, and the transmit delay that occurs at the end of each such line.

All the terminal's communications settings (those settings described in this section or in Section 11) are remembered by the terminal even when it is turned off.

All the communications settings can be set by the operator, using SETUP mode commands. See the 4113 Operator's Manual for details.

THE MOST IMPORTANT COMMUNICATIONS SETTINGS

DATA RATE COMMANDS

<Set-Baud-Rates> Command

You can set the 4113's host-to-terminal and terminal-to-host data transmission rates (also known as "baud rates"). Normally, these rates should be set by the terminal's operator, using the SETUP mode BAUD command. However, an "escape sequence" <set-baud-rates> command does exist.

CAUTION

It is usually unwise to issue the <set-baud-rates> command from the host computer. (The terminal cannot understand the command unless its baud rates are already set correctly for communicating with the host; changing the baud rates could only serve to make further communication with the host impossible.)

The <set-baud-rates> command, may, however, be included in a file of terminal commands stored in the terminal's optional disk drives. That way, the operator can initialize the terminal for operation with a particular host computer by typing a LOAD command (in SETUP mode) to load that command file into the terminal.

The <set-baud-rates> command has this syntax:

<set-baud-rates> = (ESC)(N)(R)<int><int>

(For more details, see the <set-baud-rates> command description in the 4110 Series Command Reference Manual.)

<Set-Transmit-Rate-Limit> Command

You can specify a "transmit data rate limit" — an effective maximum speed for terminal-to-host communications, which may be less than the rate at which the terminal sends each individual character. A transmit data rate limit of 300, for instance, means that the terminal, in sending characters to the host, will space those characters apart for an effective average data rate of 300 bits/second. This is useful at high baud rates, where the host computer's input processor may not be able to accept characters "back to back" at the full data rate.

For instance, even though the terminal may be set to transmit each character at, say, 9600 bits/second, it can space its characters apart for an average data transmission rate of only 300 bits/second.

The <set-transmit-rate-limit> command has this syntax:

<set-transmit-rate-limit> = (ESC)(N)(X)<int>

Here, the <int> parameter specifies the maximum effective transmit rate in bits per second.

For more details, see the description of the <set-transmit-rate-limit> command in the 4110 Series Command Reference Manual.

Examples

Table 10-1 shows examples of the <set-baud-rates> and <set-transmit-rate-limit> commands.

Table 10-1

SETTING THE TERMINAL'S DATA RATES

Example	Description
<set-baud-rates: 1200, 1200> = (ESC)(N)(R)<int: 1200><int:1200> = (ESC)(N)(R) (A)(K)(0) (A)(K)(0)	Sets the terminal's transmit (terminal-to-host) and receive (host-to-terminal) data rates to 1200 bits/second. (For information on how the <int: 1200> parameters expand to (A)(K)(0), see the description of the <int> and <int+ > parameter types in the 4110 Series Command Reference Manual.)
<set-baud-rates : 1200, 600 > = (ESC)(N)(R)<int: 1200><int: 600> = (ESC)(N)(R) (A)(K)(0) (e)(8)	Sets the terminal's transmit (terminal-to-host) rate to 1200 bits/second, and its receive rate to 600 bits/second.
<set-xmt-limit: 300> = (ESC)(N)(X)<int: 300> = (ESC)(N)(X) (R)(<)	Although the 4113 sends each character at the rate specified in the most recent <set-baud-rates> command, it spaces the characters apart for an effective average rate of 300 bits/second (about 30 characters/second).

<SET-ECHO> COMMAND

Except in LOCAL mode or SETUP mode, when you type on the 4113's keyboard, the characters typed go to the host computer. They do not necessarily appear on the terminal's screen. They only appear on the screen if (a) the host (or modem) sends the same characters back to the terminal — provides a "remote echo" — or (b) the terminal provides its own "local echo" of the transmitted characters.

The operator can specify whether the terminal provides a local echo by means of the ECHO command in SETUP mode; see the 4113 Operator's Manual for details. This command can also be issued by the host, or by a command file <load>ed from the terminal's optional disk drive. The command syntax is as follows:

<set-echo> = (ESC)(K)(E)<int>

In this command, if the <int> is 1, the terminal provides its own echo; if the <int> is zero, the terminal does not. In the latter case, it is the responsibility of the host computer (or other external equipment) to provide the echo.

Table 10-2 shows examples of the <echo> command. For more details, see the command description in Section 10.

<SET-PARITY> COMMAND

The 4113's parity setting controls how the 4113 sets the eighth bit (parity bit) in each character it sends to the host. The 4113 ignores the parity bit in characters it receives from the host.)

The choices on the use of the eighth bit are:

- Low parity. When the 4113 sends a character to the host, it sets the parity bit to zero.
- Odd parity. When sending a character to the host, the 4113 sets the parity bit so that there are an odd number of ones in the character's eight bits.
- Even parity. When sending a character to the host, the 4113 sets the parity bit so that there are an even number of ones in the character's eight data bits.
- High parity. Sets the parity bit to one in characters it transmits to the host.
- "Data" parity. The parity bit is used for data, just as are the other seven bits in each eight-bit character. (This mode is not normally used, since it implies a different code than the standard ASCII seven-bit code.)

The host computer controls the terminal's parity setting with the <set-parity> command:

<set-parity> = (ESC)(N)(P)<int>

where the <int> is 0, 1, 2, 3, or 4 for "low parity," "odd parity," "even parity," "high parity," and "parity bit used for data," respectively.

Table 10-2

SETTING THE TERMINAL'S LOCAL ECHO

Example	Description
<echo: 0> = (ESC)(K)(E)<int: 0> = (ESC)(K)(E) (0)	Specifies "no local echo." Any echo must be provided by the modem or the host computer.
<echo: 1> = (ESC)(K)(E)<int: 1> = (ESC)(K)(E) (1)	Specifies "local echo." The 4113 displays the characters it sends to the host.

<SET-STOP-BITS> COMMAND

In communicating with the host, the terminal sends and receives each character serially, as a sequence of 10 or 11 bits. (This is called "asynchronous serial" data communications.) The first bit for each character is a *start* bit, always a zero (or "space") bit. The next seven bits determine the particular ASCII character, after which comes a parity bit, described earlier. The character ends with one or two "stop bits," which are always ones (or "mark" bits). The communications line then remains in the marking condition until the start bit for the next character.

In receiving characters from the host, the terminal will always operate correctly, regardless of whether the host sends one or two stop bits in each character.

The terminal includes one or two stop bits in each character it transmits to the host. The number of stop bits can be set with the <set-stop-bits> command:

<set-stop-bits> = (ESC)(N)(B)<int>

Here, the <int> parameter specifies the number of stop bits; it must be either one or two.

The operator can also set this parameter, with the SETUP mode STOPBITS command. See the 4113 Operator's Manual for details.

LESS IMPORTANT COMMUNICATIONS SETTINGS

<SET-BREAK-TIME> COMMAND

Pressing the BREAK key sends a "break" signal to the host computer. This signal is not an ASCII character, but it is a signal to the host computer. In full duplex communications, the break is sent by holding the communications line in the "space" condition for longer than the duration of a single ASCII character. In half duplex with supervisor mode (described in Section 11), a break is sent by turning off the secondary carrier.

As shipped from the factory, the break signal is set to last 200 milliseconds. This is adequate for most host computers. The <set-break-time> command lets you change this value for use with hosts for which 200 milliseconds is too short or too long. The command has the following syntax:

<set-break-time> = (ESC)(N)(K)<int>

Here, the <int> parameter specifies the number of milliseconds that the break signal is to last. This value can also be set by the operator, using the SETUP mode BREAKTIME command; see the 4113 Operator's Manual for details. Like all communications settings, the break time is remembered by the terminal even when it is turned off.

Some host computers are intolerant of break signals. (They may, for instance, respond to a break by logging the user off and disconnecting the telephone line.) For such hosts, it may be convenient to set the break time to zero; this causes the terminal not to send a break signal.

Besides sending the break signal, the BREAK key has a few other effects. For details, see the description of the BREAK key in the 4110 Series Command Reference Manual.

COPING WITH (DEL) FILLER CHARACTERS

The Problem

Some host computers intersperse (DEL) characters (also known as (RUBOUT) characters) among the characters they send to a terminal. These extra "filler" characters are inserted automatically by the host operating system, so that the user's applications program has no control over them. Since the 4113 interprets (DEL) as a valid character in <int>, <int+>, and <xy> parameters, these extra (DEL) characters can cause problems.

The Remedy

The 4113 terminal includes two features which help you cope with this problem. Firstly, the terminal accepts the character sequence (ESC)(?) as a synonym for (DEL). Secondly, the <ignore-delete> command causes the terminal to ignore any (DEL) characters which the host may send it. (It does not, however, ignore (ESC)(?) sequences.)

Thus, if your host uses (DEL) as a filler character, you should do the following two things:

1. Write your device driver routines so that they always send (ESC)(?) when they would otherwise send the (DEL) character. The routines to change are the ones which issue <int>, <int+>, and <xy> parameters.
2. At the start of each applications program, send an <ignore-deletes> command to the terminal.

For more information on the <ignore-deletes> command, see its description in the 4110 Series Command Reference Manual.

FULL DUPLEX DATA COMMUNICATIONS

Many time-sharing systems use the "full duplex, remote echo" data communications protocol. "Full duplex" means that the data communications line can simultaneously support transmissions in both directions. "Remote echo" means that the host "echoes" back to the terminal each character which the terminal sends. It is the echo, rather than the original transmitted character, which is displayed on the terminal's screen. An echo from the host computer is possible only in a full duplex system.

If the 4113 is not equipped with Option 01, then it is always configured for full duplex data communications. To use full duplex, remote echo data communications, it is only necessary that the terminal's local echo be turned off (with a <set-echo: 0> command).

If the 4113 has Option 01 installed, then it can be used with either full duplex or half duplex data communications. In that case, a <set-duplex: 0> command is used to set the terminal for full duplex operation. The <set-duplex> command is described in Section 11.

THE COMMUNICATIONS INPUT QUEUE PROTOCOLS AND "HANDSHAKING"

<SET-QUEUE-SIZE> COMMAND

The terminal has an input buffer, or queue, where it accumulates the characters it receives from the host computer. When characters arrive faster than the terminal can process them, the terminal stores them in its communications input queue until it has a chance to process them — or until the memory allocated for that queue is exhausted. (If the queue memory is exhausted, incoming characters are lost.)

For instance, the terminal cannot display characters while it is in the process of erasing its screen. Therefore, while the screen is being erased (as, for instance, in response to the PAGE key), any characters coming from the host are stored in the input queue until the erase operation is finished. When the screen erasure is complete, the terminal reads the characters from the queue and displays them.

Again, while in SETUP mode the terminal does not display characters coming from the host. Instead, such characters accumulate in the input queue. The terminal waits to process those characters until the operator presses the SETUP key to remove the terminal from SETUP mode. The same is true for LOCAL mode.

As shipped from the factory, the communications queue can hold up to 300 ASCII characters. However, this value can be changed with the <set-queue-size> command:

<set-queue-size> = (ESC)(N)(Q)<int+>

Here, the <int+> parameter specifies how many characters the input queue can hold.

The <set-queue-size> command can also be typed by the operator, as the SETUP mode QUEUESIZE command. See the 4113 Operator's Manual for details.

THE NEED FOR HANDSHAKING

The 4113 can display simple alphanumeric text and graphics only up to a maximum continuous data rate of 9600 baud (9600 bits per second). At higher data rates, or for more complex operations, some "handshaking" protocol should be used to prevent the terminal's communications input queue from overflowing.

Even at slow data rates, it may be prudent to use a handshaking protocol. The terminal can take an appreciable amount of time to execute some commands which can be issued using only a very few characters. Examples are the <include-copy-of-segment>, <directory>, <load>, and <save> commands. If a handshaking protocol is not used, the terminal's input queue may overflow while executing such commands.

Such a handshaking protocol might be as simple as issuing <report-4010-status> from time to time, and waiting to receive the reply before sending more characters to the terminal. Alternatively, either of two data communications protocols may be used: flagging mode or block mode. (Or both modes may be used at the same time.) Either of these communications protocols will prevent the terminal's input queue from overflowing. (Block mode is available only if the terminal is equipped with Option 01. It is described in Section 11.)

<SET-FLAGGING-MODE> COMMAND

The terminal's flagging mode can be set by the operator using the SETUP mode FLAGGING command. (See the 4113 Operator's Manual for details.) It can also be set by the host computer, using the <set-flagging-mode> command:

<set-flagging-mode> = (ESC)(N)(F)<int>

Here, the <int> parameter specifies the flagging mode; it is in the range from 0 to 4.

Mode 0 (No Flagging). DC1/DC3 and DTR/CTS flagging are both disabled.

Mode 1 (DC1/DC3 Flagging for Input). The terminal uses the "DC1/DC3" flagging protocol when receiving characters from the host.

If the host is sending characters to the terminal faster than the terminal can process them, so that the terminal's input queue is in danger of overflowing, then the terminal sends the host a (DC3) character. The host is then expected to suspend transmission of characters to the terminal.

When the terminal is ready for more characters, it sends the host a (DC1) character. This signals the host that it may resume transmission of characters to the terminal.

Mode 2 (DC1/DC3 Flagging for Output). The terminal uses the "DC1/DC3" flagging protocol when transmitting characters to the host.

The host can send the terminal a (DC3) character when the host's input buffer is in danger of overflowing. The 4113 sends at most one or two more characters and then stops transmitting to the host.

When the host is ready for more characters, it sends the terminal a (DC1) character. When the 4113 receives a (DC1), it resumes transmission to the host.

Mode 3 (DC1/DC3 Flagging for Both Input and Output). The terminal uses the DC1/DC3 flagging protocol both when receiving characters from the host and when transmitting characters to the host.

Mode 4 (DTR/CTS Flagging). In DTR/CTS flagging, the terminal uses two signal lines at the RS-232 connector to regulate the flow of data between the terminal and the host computer. These signal lines are DTR (Data Terminal Ready) and CTS (Clear To Send).

NOTE

DTR/CTS flagging is usually not practical when the host is connected to the terminal over telephone lines by the use of modems. In such circumstances, the host does not have direct access to the DTR and CTS signal lines. This flagging mode is only practical if the host is connected directly to the terminal.

In DTR/CTS flagging, the terminal indicates that it wishes to transmit data by asserting RTS (Request To Send); that is, it places a positive voltage on the RTS signal line at the RS-232 connector. If the host is ready to receive the data, it asserts CTS (Clear To Send). The terminal is only allowed to transmit when CTS is asserted.

Should the terminal be transmitting characters faster than the host can process them, so that the host's input buffer is in danger of overflowing, the host can drop CTS (place a negative voltage on the CTS signal line). When the host is ready to receive more characters, it asserts CTS again, and the terminal resumes its transmission.

When receiving characters from the host, the terminal uses the DTR (Data Terminal Ready) signal line in the same way that the host uses the CTS line. If the host is sending characters faster than the terminal can process them, so that the terminal's input queue is in danger of overflowing, then the terminal drops DTR. (It places a negative voltage on the DTR signal line.) The host is then expected to stop transmitting to the terminal. When the terminal is ready for more characters, it asserts DTR (places a positive voltage on the DTR line), and the host resumes its transmission to the terminal.

PROMPT MODE

Besides flagging mode (just described) and block mode (described in Section 11), the terminal has a third handshaking protocol: prompt mode.

The prompt mode protocol is useful for preventing the host's input buffer from overflowing when the terminal has much data to send to the host. However, prompt mode does not protect the terminal's input queue from overflowing when the host sends data to the terminal. For that, you should use flagging mode or block mode.

Prompt Mode Operation

When the terminal is in prompt mode, it waits to send each line of text until it receives a *prompt string* from the host computer. (The prompt string is a sequence of ASCII characters, determined by the most recent <set-prompt-string> command.)

On receiving the prompt string, the terminal waits for the "transmit delay" amount of time and then sends one line of text to the host computer. Here, "one line of text" means all the characters it has to send, up to and including the next <EOM-char> or <EOL-string>. This is described in more detail later in this section. Typically, this means all characters up to and including the next carriage return (CR) character.

The prompt string must be the last characters received by the terminal, or else the terminal will not recognize it as a prompt.

The following steps summarize prompt mode operation:

1. The terminal sends a line of text, up to and including the <EOM-char> or <EOL-string> that marks the end of the line.
2. The terminal receives a prompt string from the host.
3. The terminal waits for the transmit delay.
4. Steps 1 through 3 are repeated again and again, until the terminal is removed from prompt mode.

End of a Line of Text. As just mentioned, the end of a line of text occurs when the terminal encounters an <EOM-char> or <EOL-string> in the data it is sending to the host.

Typically, an <EOM-char> is a (CR) chracter typed by the operator or occurring in a file being transferred to the host with the <copy> command. (The <copy> command is described in Section 12.) However, other characters can be chosen as the <EOM-char>. For details, see the description of the <set-EOM-chars> command in the 4110 Series Command Reference Manual.

If the terminal is not in block mode, then it sends <EOL-string>s wherever <EOM-indicator> flags occur in a report it is sending to the host computer. (Examples of such reports are <GIN-report-sequence>s and <terminal-settings-report>s. See Sections 8 and 9 for details; see also the Command Reference Manual for the syntax of the different report messages.) The <EOL-string>s are typically (CR) characters, although other characters or character sequences can be chosen with the <set-EOL-string> command.

If the terminal is in block mode, it sends an <EOL-string> at the end of each line of a block being sent to the host. (Block mode is described in Section 11.)

<Prompt-Mode> Command

You can put the terminal in prompt mode or remove it from prompt mode with the <prompt-mode> command:

<prompt-mode> = (ESC)(N)(M)<int>

If the <int> parameter is zero, the terminal exits prompt mode. (It is not an error to send a <prompt-mode: 0> command when the terminal is already out of prompt mode.)

If the <int> parameter is two, the terminal enters prompt mode as soon as it processes the <prompt-mode: 2> command. Even if the terminal is in the midst of sending characters to the host, it stops sending those characters. (It waits for a prompt from the host before resuming transmission.)

If the <int> parameter is one, and the terminal is sending characters to the host, then it finishes sending the current line of text before entering prompt mode. That is, the terminal continues to transmit characters until it encounters an <EOM-char> or <EOL-string> in the data being sent to the host. After sending that <EOM-char> or <EOL-string> — typically, the (CR) character — the terminal enters prompt mode. Thereafter, it will not send more characters until the transmit delay elapses and it receives a prompt from the host.

The operator can remove the terminal from prompt mode by typing the SETUP mode command, PROMPT-MODE NO, or by pressing the CANCEL key. The host can remove the terminal from prompt mode by sending the <cancel> command or the <prompt-mode: 0> command.

<Set-Prompt-String> Command

The prompt string is determined by the <set-prompt-string> command:

<set-prompt-string> = (ESC)(N)(S)<int-array>

Here, the <int-array> is an array of up to ten integers. Each integer in the array is the numeric equivalent of an ASCII character.

For instance, to set the prompt string to "GIMME:," you can send the following command to the terminal:

<set-prompt-string: "GIMME:">
= (ESC)(N)(S)
  <int-array: (71,73,77,77,69,58)>
= (ESC)(N)(S)
  <int: 6>
  <int: 71><int: 73><int: 77>
  <int: 77><int: 69><int: 58>
= (ESC)(N)(S) (6) (D)(7) (D)(9)
  (D)(=) (D)(=) (D)(5) (C)(:)

Here, the integer 6 is the number of <int>s in the <int-array>. The integers 71, 73, 77, 77, 69, and 58 are the decimal equivalents of the ASCII characters (G), (I), (M), (M), (E), and (:).

LINES OF TEXT AND THE TRANSMIT DELAY

The preceding description of prompt mode mentioned the concept of a "line of text" in data being sent to the host, and the transmit delay which occurs after the terminal sends each such line of text. The following description explains these in more detail.

THE TRANSMIT BUFFER

The terminal has an internal transmit buffer to hold any characters which are waiting to be sent to the host computer. When the operator types on the keyboard, the characters he or she types go into the transmit buffer. Likewise, when the terminal has a report message to send to the host, the characters of that message go into the transmit buffer.

The terminal sends the characters in the transmit buffer to the host a line at a time; that is, it reads characters from the transmit buffer and sends them to the host until it encounters the end of a line of text. Then it waits for a short time to elapse (the "transmit delay") before sending the next line of text. If in prompt mode, the terminal also waits to receive a prompt from the host before sending the next line of text.

For this purpose, a "line of text" means "all the characters waiting to be transmitted, up to and including the next <EOM-char> or <EOL-string>." As the terminal is shipped from the factory, its only <EOM-char> is (CR), and the <EOL-string> consists of one character, (CR). Thus, if the terminal is set as it is when shipped from the factory, then a "line of text" means "all characters to be transmitted, up to and including the next (CR) character."

If there are no characters waiting in the transmit buffer, then each character the operator types enters the transmit buffer and is immediately sent to the host computer. (This is true even if the terminal is in prompt mode.) Only if the operator could type faster than the terminal's transmit rate would more than one character be in the transmit buffer. When the operator presses RETURN, a (CR) goes into the transmit buffer and is sent to the host. Since (CR) is the usual <EOM-char>, this also marks the end of a line of text. The terminal then waits a short time before sending the next character typed. This transmit delay, however, is usually so short as to be imperceptible to the operator. (However, in half duplex mode — described in Section 11 — the transmit delay does last long enough to give the host a chance to seize the communications line and send a message to the terminal.)

<SET-TRANSMIT-DELAY> COMMAND

The <set-transmit-delay> command has this syntax:

<set-transmit-delay> = (ESC)(N)(D)<int>

Here, the <int> parameter specifies the approximate number of milliseconds in the transmit delay. The operator can also change this setting by means of the SETUP mode XMTDELAY command. As shipped from the factory, the delay is set to 100 milliseconds.

Section 11

OPTION 01: HALF DUPLEX AND BLOCK MODE

This section describes the half duplex and block mode data communications protocols, which are available if the 4113 has Option 01 installed.

With Option 01, the terminal supports three types of half duplex communications protocol, which are discussed in this section along with full duplex mode. Option 01 also provides block mode protocol, which is a method of formatting data for transmission to and from the host.

FULL AND HALF DUPLEX DATA COMMUNICATIONS

<SET-DUPLEX-MODE> COMMAND

Terminals not equipped with Option 01 are always set to use the full duplex communications protocol. If the terminal has Option 01 (Half Duplex and Block Mode) installed, then you can select the duplex mode with the <set-duplex-mode> command:

<set-duplex-mode> = (ESC)(O)(D)<int>

Here the <int> parameter is zero for full duplex mode, one for half duplex normal, two for half duplex with automatic request to send, or three for half duplex with supervisor mode.

Normally, however, the operator (rather than the host computer) selects the duplex mode. To do this, the operator uses the SETUP mode command, DUPLEX. See the 4113 Operator's Manual for details.

FULL DUPLEX MODE

In full duplex mode, the data communications line can simultaneously carry data to and from the host computer.

In full duplex mode, the terminal sends break signals by sending a "space" for longer than the duration of a single ASCII character. That is, the terminal puts a positive voltage on the TDATA (transmitted data) line at the RS-232 connector for a period of time determined by the <set-break-time> command. (See Section 10 for a description of the <set-break-time> command. See also the description of that command in the 4110 Series Command Reference Manual.)

If Option 01 is installed, the operator can select full duplex mode with the SETUP mode command, DUPLEX FULL. There is also an escape-sequence version of this command, <set-duplex-mode: 0>, as follows:

<set-duplex-mode: 0> = (ESC)(O)(D)<int: 0>
= (ESC)(O)(D)(0)

If Option 01 is not installed, the terminal is always set to operate in full duplex mode.

HALF DUPLEX DATA COMMUNICATION

Some computer systems use the "half duplex" method of data communication, in which the terminal and the host computer take turns using the same data communications channel. Half duplex mode offers faster data rates than full duplex.[1] For the faster data rates, however, there is a price: it is not possible in a half duplex system for the host computer to provide a "remote echo" of data which the terminal sends.

With Option 01 installed, the 4113 supports three variations of the half duplex communications protocol: half duplex normal, half duplex with automatic request to send, and half duplex with supervisor. These protocols differ primarily in how they "turn the line around."

Unfortunately, even systems programmers often are not aware of the subtle differences between the various half duplex protocols. This makes it difficult to get the information you need to set the terminal correctly. For instance, you might be told that your host computer uses "half duplex" protocol, when in fact it uses "full duplex," but with the terminal required to provide its own local echo.

[1] For simple modems using frequency-shift-keying over voice-grade telephone lines, full duplex modems typically run at a maximum of 300 bits/second, while half duplex modems run at up to 1200 bits/second. Higher data rates are possible with more sophisticated modems or higher-quality communications lines; in general, however, the half duplex modems still offer higher data rates than comparable full duplex modems.

Half Duplex Normal

In half duplex normal mode, there is only one communications channel. The terminal and the host computer take turns using this one communications channel.

Host to Terminal. When the host transmits to the terminal, it seizes the communications line. It does this by sending the RTS — Request To Send — signal to its modem. This causes the modem to place a carrier tone on the telephone line.

While the host has control of the line, the terminal cannot transmit data to the host. (The terminal's modem detects the carrier tone, and sends a DCD — Data Carrier Detect — signal to the terminal. While the DCD signal is present, the terminal is inhibited from transmitting.) If the terminal has data to send, it stores that data in an internal transmit queue, until the host finishes its transmission and releases the line.

Terminal to Host. Suppose the host is not using the communications line. Then, as soon as the terminal has data to send, the terminal seizes the line. (As soon as the operator types the first character of a message, the terminal asserts RTS, causing its modem to send a carrier tone over the telephone line.)

The terminal continues to hold the line (continues to assert RTS) until it encounters an <EOM-char> or <EOL-string> in the data which it sends to the host. The <EOM-char> or <EOL-string> marks the end of the "line of text." The terminal sends the <EOM-char> or <EOL-string> and then pauses for a short time. (This is the "transmit delay," described in Section 10.)

During the transmit delay, the terminal relinquishes control of the communications line. (It turns off the RTS signal, causing its modem to stop transmitting the data carrier tone.) This gives the host computer a chance to size the communications line and send data to the terminal.

NOTE

For half duplex normal mode (or any half duplex mode) to work properly, it is important that the transmit delay should be set to a sufficiently long time for the host to respond and seize the communications line.

If the terminal has more data to send and the transmit delay time expires without the host's seizing the line, then the terminal seizes the line again (asserts RTS again) and sends another sequence of characters to the host. As before, the terminal continues to hold the line until it reaches the end of the line. (That is, the terminal continues to assert RTS until it has transmitted the <EOM-char> or <EOL-string> marking the end of the line of text.)

For more information on <EOM-char>s and <EOL-string>s, see "Lines of Text and the Transmit Delay" in Section 10. See also the 4110 Series Command Reference Manual for descriptions of <EOM-char>, <EOM-indicator>, <EOL-string>, <set-EOM-chars>, and <set-EOL-string>.

Selecting Half Duplex Normal Mode. The operator can place the terminal in half duplex normal mode with the SETUP mode command, DUPLEX NORMAL. (See the 4113 Operator's Manual for details.) There is also an escape sequence version of this command, <set-duplex-mode: 1>, as follows:

<set-duplex-mode: 1> = (ESC)(O)(D)<int: 1>
= (ESC)(O)(D)(1)

Half Duplex with Automatic Request to Send

The *half duplex with automatic request to send* mode differs only slightly from half duplex normal mode.

In half duplex normal mode, if neither the host nor the terminal has any data to send, then neither one seizes the communications line. (Neither host nor terminal asserts RTS, so neither modem sends a carrier tone over the telephone line.)

Some computers, however, will disconnect from the telephone line if there is no carrier tone being sent over that line.

The *half duplex with automatic request to send* mode is provided for use with such host computers. In this mode, the terminal seizes the line all the time (asserts RTS all the time), except only for the short transmit delay time which occurs at the end of each line of text sent to the host. If the host does not seize the line during that transmit delay, then the terminal will seize the line again (assert RTS again). The terminal does this whether or not it has any data to send.

The operator can place the terminal in half duplex with automatic request to send mode by typing the SETUP mode command, DUPLEX ARTS. There is also an escape sequence command, <set-duplex: 2>, as follows:

<set-duplex: 2> = (ESC)(O)(D)<int:2>
= (ESC)(O)(D)(2)

Half Duplex with Supervisor

In half duplex with supervisor mode there is, besides the main data communications channel, a secondary slow-speed channel. This secondary channel is called the "supervisory channel," and the host computer can use it to control line turnaround operations.

Terminal to Host. When the terminal is transmitting data to the host, the host sends a signal on the secondary channel. (The host sends an SRTS — Secondary Request To Send — signal to its modem; this causes the modem to place a "secondary carrier" on the telephone line.) The secondary carrier serves to notify the 4113 that the host is listening.

As with the other half duplex modes, the terminal pauses at the end of each line, dropping RTS (Request To Send) for the duration of the transmit delay. As with the other half duplex modes, this pause is intended to give the host a chance to seize the communications line and send data back to the terminal.

Unlike the other half duplex modes, however, the half duplex with supervisor mode allows the host to seize the communications line at *any* time — not just during the transmit delay at the end of each line. To seize the line, the host stops sending the SRTS (Secondary Request To Send) signal. This causes its modem to stop transmitting the secondary carrier. When the terminal's modem detects this, it turns off the SDCD (Secondary Data Carrier Detect) signal at the terminal's RS-232 connector. The terminal is then obliged to relinquish control of the communications line. The terminal may send no more than two characters before relinquishing control and turning off its RTS signal.

Host to Terminal. When the host transmits data to the terminal, it asserts RTS, causing its modem to send a data carrier tone to the terminal. The terminal's modem, on receiving the carrier, notifies the terminal of this by asserting the DCD (Data Carrier Detect) signal at the RS-232 connector. The terminal asserts SRTS, causing its modem to send the secondary carrier signal to the host's modem. This notifies the host that the terminal is to receive data.

The terminal cannot transmit to the host until the host relinquishes control of the data communications line. That is, it cannot transmit until the host turns off the data carrier (stops sending the RTS signal). This would normally happen when the host is done transmitting data to the terminal.

While the host is transmitting to the terminal, the terminal's operator can, if he wishes, press the BREAK key. (He would do this if he wanted to interrupt the host.) Pressing BREAK causes the terminal to drop the secondary carrier (stop asserting SRTS) for a short period of time. (This time can be set by the <set-break-time> command.) By ceasing to send the secondary carrier, the terminal is, in effect, telling the host that the terminal would like a chance to use the communications line.

However, the host computer need not honor the "break" request. It can just keep transmitting. In half duplex supervisor mode, if the host sends a "break" (drops the secondary carrier), the terminal must relinquish the line. But if the terminal sends a "break," the host need not relinquish the line.

Selecting Half Duplex With Supervisor Mode. The operator can put the terminal in half duplex with supervisor mode by typing the SETUP mode command, DUPLEX SUPER. There is also an escape-sequence version of this command, <set-duplex-mode: 3>, as follows:

<set-duplex-mode: 3> = (ESC)(Q)(D)<int: 3>
= (ESC)(O)(D)(3)

BLOCK MODE

INTRODUCTION

The terminal's block mode protocol is provided as part of Option 01. Block mode is a method of formatting data when sending it to and from the host.

Some host operating systems make it difficult for the user's program to send or receive the full ASCII character set. For instance, the host may reject certain control characters, such as (ESC). The host may even convert lowercase characters to uppercase ones.

The terminal's block mode communications protocol provides a way to cope with such hosts. In block mode, messages using the full ASCII character set are packed into character strings using a subset of that character set.

Also, the block mode protocol provides error detection and automatic retransmission of bad data blocks. This lets you transfer data to and from the terminal, without errors, despite occasional noise on the communications line.

Block mode is completely independent of whether or not prompt mode is used and of whether the terminal is using full duplex or half duplex communications.

When in block mode, data is packed into blocks and transmitted as a unit. The host must send the first block. The terminal will send a block only in response to the host.

Each block contains an "even/odd" counter (block control byte one, bit one) which is used in an "ACK/NAK" protocol. This protocol lets the host and terminal inform each other when a block has been received incorrectly. (The block received incorrectly is then retransmitted.)

BLOCK FORMAT

Overall Syntax

Blocks sent from the host and from the terminal have the same overall format. Each block consists of one or more lines of data, as follows:

```
<block> = [<block-"other-than-last"-line> . . .]
          <block-last-line>
```

where

```
<block-"other-than-last"-line>
= <block-header>
  <block-packed-data>
  <block-continue-char>
  <EOL-string>
```

and

```
<block-last-line> = <block-header>
                    <block-packed-data>
                    <block-end-char>
                    <EOL-string>
```

Each line begins with a special <block-header> character sequence, which is set by the <set-block-headers> command. After the header comes the packed data, followed by a special character to mark the end of the line. This is the <block-end-char> for the last (or only) line of a block, or the <block-continue-char> if the line is not the last line of the block. After the <block-end-char> comes an end-of-line string. For blocks sent from the terminal to the host, this is the terminal's current <EOL-string>, as set by the most recent <set-EOL-string> command. For blocks sent from the host to the terminal, the end-of-line-string can be any sequence of characters.

As the terminal is shipped from the factory, the header strings are "HEADRX" for blocks sent from the host to the terminal, and "HEADTX" for blocks sent from the terminal to the host. The <block-continue-char>s for transmission to the host and to the terminal are both (&). The <block-end-char>s are both ($). The <EOL-string> for transmission to the host is the single character, (CR). All these values can be changed by commands to the terminal.

Blocks Sent From the Host to the Terminal

Using these default settings, a single-line block from the host to the terminal might take the following form:

(H)(E)(A)(D)(R)(X)<block-packed-data>($)(CR)(LF)

(Here, it is assumed that the host ends each line with the character sequence (CR)(LF).)

Likewise, a three-line block from the host to the terminal would take this form:

(H)(E)(A)(D)(R)(X)<packed-data>(&)(CR)(LF)
(H)(E)(A)(D)(R)(X)<packed-data>(&)(CR)(LF)
(H)(E)(A)(D)(R)(X)<packed-data>($)(CR)(LF)

Blocks Sent From the Terminal to the Host

Assuming that the terminal's <EOL-string>, as set by the most recent <set-EOL-string> command, is the single (CR) character, then a three-line block from the terminal to the host would take this form:

(H)(E)(A)(D)(T)(X)<packed-data>(&)(CR)
(H)(E)(A)(D)(T)(X)<packed-data>(&)(CR)
(H)(E)(A)(D)(T)(X)<packed-data>($)(CR)

In these examples, the <block-header>s, <block-continue-char>s, and <block-end-char>s all have their default settings. These settings can be changed on command. The settings for blocks sent to the terminal can be different from those for blocks sent from the terminal to the host. See the 4110 Series Command Reference Manual for descriptions of the <set-block-headers>, <set-block-continue-chars>, and <set-block-end-chars> commands.

ENTERING AND LEAVING BLOCK MODE

Entering Block Mode

The procedure for putting the terminal in block mode is as follows:

1. First, the host computer or the terminal's operator issues commands to set the various block mode parameters. (These commands must be sent *before* the terminal is armed for block mode.)

Escape Sequence Command (From Host Computer)	SETUP Mode Command (From Operator)
<set-block-continue-chars>	BCONTINUECHARS
<set-block-end-chars>	BENDCHARS
<set-block-master-chars>	BMASTERCHARS
<set-block-non-xmt-chars>	BNONXMTCHARS
<set-block-headers>	BHEADERS
<set-block-length>	BLENGTH
<set-block-line-length>	BLINELENGTH
<set-block-packing>	BPACKING
<set-block-timeout>	BTIMEOUT

   (These settings are remembered by the terminal even when it is turned off.)

2. Next, the terminal is armed for block mode:

Escape Sequence Command	SETUP Mode Command
<arm-for-block-mode>	BLOCKMODE

3. Finally, the host sends a block to the terminal. The terminal enters block mode on receiving the header characters that begin the first line of that block.

Exiting Block Mode

From the Host. The host removes the terminal from block mode by sending it a special block containing an "exit protocol" command in that block's block control bytes. (The block control bytes are described later in this section. The "exit protocol" command consists of sending <block-control-byte-1> with its Bit 2 set to one rather than zero.)

From the Keyboard. The operator can remove the terminal from block mode with the SETUP mode command, BLOCKMODE NO.

MAXIMUM LINE LENGTH

There is no maximum line length for blocks sent from the host to the terminal. If you like, you can have the host send each block to the terminal as one long line. On the other hand, if you find it convenient to break the block into several lines, you can do that, too.

There is, however, a maximum length for lines which the terminal sends to the host. This value is set with the <set-block-line-length> command. (As shipped from the factory, the terminal is set for a maximum line length of 70 characters.) For this purpose, the length of a line in a block is the number of characters in the header string, plus the number of characters of packed data on that line, plus one for the <block-end-char> or <block-continue-char>. The <EOL-string> is not included in this count.

For instance, suppose that the host computer can reliably accept up to 80 characters of data on each line, not counting (CR)s as "characters of data." In that case, you can set the terminal's maximum block line length to 80. To do this, issue a <set-block-line-length> command as follows:

<set-block-line-length: 80>
= (ESC)(O)(S)<int: 80>
= (ESC)(O)(S)(E)(0)

PACKING DATA INTO A BLOCK

Packed and Unpacked Data

The "unpacked data" for a block includes the characters of the message (if any) being sent in the block, plus four more characters, the block control bytes:

<unpacked-data> = <characters-of-message>
<block-control-bytes>

The block control bytes are described later in this section. They are also described, under "<Block-Control-Bytes>," in Section 4 of the 4110 Series Command Reference Manual.

In packing the data into the block, the <unpacked-data> character sequence is transformed into another character sequence which uses a restricted subset of the ASCII character set. This character sequence is the "packed data." The transformation from unpacked data to packed data is done according to a particular packing algorithm:

<packed-data> = a sequence of ASCII characters which is produced by applying the packing algorithm to the <unpacked-data> character sequence

Maximum Block Length

The terminal has a "maximum block length" for the <unpacked-data> in blocks sent from the terminal to the host, and a another maximum block length for the <unpacked-data> in blocks sent from the host to the terminal. When the terminal is shipped from the factory, both these maximums are set to 256 characters. These values can be changed with the <set-block-length> command; see the 4110 Series Command Reference Manual for details.

Assume that the maximum block length is 256 characters. If the terminal is in block mode, and the operator types 500 characters before pressing RETURN, then the terminal splits that 500-character message into two blocks. The first block's <unpacked-data> consists of the first 252 characters that the operator typed, plus the four block control bytes. The second block holds the remaining 248 of the 500 characters, plus the (CR) character typed by pressing RETURN, plus four block control bytes: a total of 253 characters of <unpacked-data>.

Likewise, if the host has a message of more than 252 characters to send the terminal, it must split that message into more than one block. The first block sent to the terminal would have, in its <unpacked-data>, the first 252 characters of the message, plus four block control bytes for that block. The remainder of the message would go into the <unpacked-data> for subsequent blocks.

<EOM-Char>s and <EOM-Indicator>s

When the terminal is sending data to the host, it groups the data into "messages." Each message text ends with an <EOM-char> (end-of-message character) or <EOM-indicator> (end-of-message indicator).

<EOM-Char>s. Typically, the <EOM-char> is (CR). Thus, each time the operator presses RETURN, he terminates one "message." If the terminal is in block mode, the (CR) is packed into the block and causes the terminal to end that block.

<EOM-Indicator>s. If the data being sent to the host is a report message, then there are one or more <EOM-indicator>s in the syntax for that report. (The syntax for each report message type is given in the 4110 Series Command Reference Manual.) Every report has an <EOM-indicator> at its end. In addition, some reports may have optional <EOM-indicator>s embedded within their syntax to split the report into several "lines of text." Examples are the <GIN-report-sequence>, <segment-status-report>, and <error-report> message types.

In block mode, when the terminal encounters an <EOM-indicator> in a message being sent to the host, it terminates the current block and sets an "end-of-message" bit in one of the block control bytes.

The effect of <EOM-indicator>s is similar to that of <EOM-char>s. Both <EOM-indicator>s and <EOM-char>s cause the terminal to end the current block and set the end-of-message bit in the block control bytes. However, <EOM-char>s are packed into the block, while <EOM-indicator>s are not.

Non-Transmittable Characters

Certain "non-transmittable characters" are not allowed to occur within packed data. For instance, the <block-end-char> and <block-continue-char> may not occur in the packed data, for they signal the end of lines in the block.

When <unpacked-data> is packed into the block, any non-transmittable characters occuring in the <packed-data> are replaced with two-character sequences. The first character is the <block-master-char>, while the second is an uppercase letter.

To tell the terminal which characters are non-transmittable, you issue a <set-block-non-xmt-chars> command:

<set-block-non-xmt-chars>
= (ESC)(O)(N)<int-array><int-array>

The first <int-array> holds the numeric equivalents of the non-transmittable characters for blocks sent from the host to the terminal. Likewise, the second <int-array> holds numeric equivalents for the host-to-terminal non-transmittable characters. During block transmission, the first character specified in each <int-array> is replaced with <block-master-char> (A); the second character is replaced with <block-master-char> (B); and so on.

If the <block-continue-char>, <block-end-char>, or <block-master-char> could otherwise occur in <unpacked-data>, then they must be included in the set of non-transmittable characters.

When the terminal is shipped from the host, its non-transmittable characters are set to (#), ($), and (&), in that order. This is because (#) is the default <block-master-char>, ($) is the default <block-end-char>, and (&) is the default <block-continue-char>. Thus, any (#) character which otherwise would occur in the <unpacked-data> is replaced with (#)(A); any ($) occurring in the <unpacked-data> is replaced with (#)(B); and any (&) is replaced with (#)(C).

Packing Algorithm

The packing algorithm is as follows.

Step One. Examine the number of bits per unpacked data byte, and the number of bits per packed "pseudo-byte," as set by the <set-block-packing> command. If they are equal, proceed to Step Four. If they are unequal, proceed to Step Two.

Step Two. The block's <unpacked-data> is regarded as a sequence of 7-bit or 8-bit bytes laid "end to end," forming one long stream of binary bits. The first bit is the high-order bit of the first byte; the last bit is the low-order bit of the last byte.

The <set-block-packing> command determines whether the <unpacked-data> consists of 7-bit or 8-bit bytes. (The factory default setting is "7-bit bytes," since ASCII characters have seven data bits.) For details, see the <set-block-packing> description in the 4110 Series Command Reference Manual.

Step Three. Next, the bit stream is divided into a series of "pseudo-bytes" of six, seven, or eight bits each. An offset is added to each pseudo-byte, thereby converting it into a standard ASCII character. Again, it is the <set-block-packing> command which determines whether each pseudo-byte consists of six, seven, or eight bits.

Table 11-1 shows the offset which is added for each allowable pseudo-byte size.

Table 11-1

PACKED PSEUDO-BYTE CHARACTERISTICS

No. of Meaningful Data Bits per Pseudo-Byte	Offset Added To Make a Standard ASCII Character	Range of Possible Decimal Equivalents for ASCII Characters To Be Transmitted
6	32	32 to 95 ASCII characters from (SP) to (_)
7	0	0 to 127 Full ASCII character set
8	0	0 to 255 Full eight-bit data bytes

If there are not enough bits to fill out the last pseudo-byte, an appropriate number of zeroes are appended to the end of the bit stream. On input, these extra zeroes are ignored. The extra zeroes are inserted only at the end of a block — not at the end of lines (other than the last line) within the block.

Step Four. If any of the resulting characters are among the non-transmittable characters, they are replaced with two-character sequences, starting with the "master character." The first non-transmittable character specified in the <set-block-non-xmt-chars> command is replaced, wherever it occurs, with the master character followed by the letter A; the second non-transmittable character is replaced by <master-char> (B); and so on.

An Example

Suppose the following:

- The terminal is in block mode, and the block mode parameters are at their factory settings. That is, the <block-continue-char>, <block-end-char>, and <block-master-char> are (&), ($), and (#), respectively; the non-transmittable characters are (#), ($), and (&), in that order; the terminal-to-host and host-to-terminal header strings are "HEADRX" and "HEADTX," respectively; (CR) is the <EOM-char>; the <EOL-string> is the single character, (CR); and the unpacked and packed bits-per-byte settings are seven and six, respectively.
- The host has just sent an odd-numbered block to the terminal, with the end-of-message bit set.
- The terminal's operator types "BEGIN_PROGRAM" and presses RETURN.

Then the packing proceeds as follows:

1. The characters (B)(E)(G)(I)(N)(_)(P)(R)(O)(G)(R)(A)(M)(CR), plus four <block-control-bytes>, comprise the <unpacked-data> for a block. The (CR) — typed by pressing RETURN — is an <EOM-char>; as such, it signals the terminal to compute the block control bytes, pack them into the current block, and send that block to the host.

2. The <unpacked-data> consists of the text "BEGIN_PROGRAM", the (CR) character, and the four control bytes, as follows:

character:	(B)	(E)	(G)
binary:	1000010	1000101	1000111
character:	(I)	(N)	(_)
binary:	1001001	1001110	1011111
character:	(P)	(R)	(O)
binary:	1010000	1010010	1001111
character:	(G)	(R)	(A)
binary:	1000111	1010010	1000001
character:	(M)	(CR)	(A)
binary:	1001101	0001101	1000001
character:	(NUL)	(BEL)	(F)
binary:	0000000	0000111	1000110

(The last four characters — (A)(NUL)(BEL)(F) — are the <block-control-bytes>. They are described later in this section, and in the 4110 Series Command Reference Manual.)

3. The terminal regroups the stream of binary bits into six-bit pseudo-bytes, as follows:

decimal:	(33)	(17)	(24)	(60)
binary:	100001	010001	011000	111100
decimal:	(38)	(29)	(31)	(40)
binary:	101000	100110	011101	011111
decimal:	(20)	(41)	(60)	(30)
binary:	010100	101001	111100	011110
decimal:	(37)	(1)	(38)	(35)
binary:	100101	000001	100110	100011
decimal:	(24)	(8)	(0)	(15)
binary:	011000	001000	000000	001111
decimal:	(6)			
binary:	000110			

4. Since the unpacked pseudo-byte size is six, Table 11-1 calls for an offset of 32 (binary 100000) to be added to each pseudo-byte. This converts each pseudo-byte to an ASCII character in the range from (SP) to (_):

decimal:	(65)	(49)	(56)	(92)
binary:	1000001	0110001	0111000	1011100
character:	(A)	(1)	(8)	(\)
decimal:	(70)	(61)	(63)	(72)
binary:	1000110	0111101	0111111	1001000
character:	(F)	(=)	(?)	(H)
decimal:	(52)	(73)	(92)	(62)
binary:	0110100	1001001	1011100	0111110
character:	(4)	(I)	(\)	(>)
decimal:	(69)	(33)	(70)	(67)
binary:	1000101	0100001	0100110	1000011
character:	(E)	(!)	(F)	(C)
decimal:	(56)	(40)	(32)	(47)
binary:	0111000	0101000	0100000	0101111
character:	(8)	("(")	(SP)	(/)
decimal:	(38)			
binary:	0100110			
character:	(&)			

5. The <block-continue-char>, (&), occurs at the end of this sequence. Since it is the third non-transmittable character, it is replaced by <block-master-char> (C), or (#)(C). This gives the following sequence of characters holding the packed data:

   <packed-data> = (A)(1)(8)(\)(F)(=)(?)(H)(4)(I)(\)(>)(E)(!)(F)(C)(8)("(")(SP)(/)(#)(C)

6. The terminal composes and sends a one-line block using this <packed-data>:

   <block> = <block-header>
   <packed-data>
   <block-end-char>
   <EOL-string.

   = (H)(E)(A)(D)(T)(X)
   (A)(1)(8)(\)(F)(=)(?)(H)(4)(I)(\)
   (>)(E)(!)(F)(C)(8)("(")(SP)(/)(#)(C)
   ($)
   (CR)

   (Actually, the block is composed and transmitted "on the fly" — character by character — as the operator types the data on the keyboard. When the operator presses RETURN, the (CR) and block control bytes are packed and sent. Then the final characters of the block are sent: the <block-end-char> and <EOL-string>.)

THE BLOCK CONTROL BYTES

In block mode, when the terminal or host composes a block to be sent over the data communications line, it appends four "block control bytes" to the characters or other data being packed into the block:

<unpacked-data> = <characters-of-message>
<block-control-bytes>

<block-control-bytes> = <control-byte-1>
<control-byte-2>
<control-byte-3>
<control-byte-4>

The four control bytes are packed into the block along with the other unpacked data. If the "unpacked byte size" (as set by the <set-block-packing> command) is seven, then each control byte consists of seven binary bits. If the unpacked byte size is eight, then each control byte consists of eight binary bits.

Every block contains at least the four block control bytes, even if it contains no other characters of <unpacked-data>.

<Control-Byte-1>

Let Bit 1 be the least-significant bit of the byte; then Bit 7 or Bit 8 is the most-significant bit of the byte. (Bit 7 is the most-significant bit if the "unpacked byte size" is seven, since in that case there is no Bit 8.) The individual bits are assigned as follows:

- Bits 1 and 2: Block count and end-protocol.
- Bit 3, 4, 5: Reserved (always zero)
- Bit 6: End of file
- Bit 7: End of message
- Bit 8: Unused (not present in 7-bit bytes; always zero in 8-bit bytes)

Bits 1 and 2. In <control-byte-1>, Bits 1 and 2 together serve two functions: they determine whether the terminal is to exit block mode, and — while the terminal is in block mode — they maintain an "odd/even" modulo two counter of blocks sent over the data communications line. Table 11-2 lists the four possible states of these bits, together with their meanings.

End-of-File Bit. Bit 6, the end-of-file bit, is set to one at the end of a file transfer; setting this bit serves the same purpose as the "end-of-file string" used when the terminal is not in block mode.

End-of-Message Bit. In blocks which the terminal sends to the host, Bit 7, the end-of-message bit, when set to one, indicates that the terminal has terminated the block because it encountered an <EOM-char> or <EOM-indicator> in the data being sent. When set to zero, this bit indicates that the block was terminated only because the maximum block length was reached, and that another block follows which contains more of the same message.

In blocks sent from the host to the terminal, Bit 7 has a different meaning. If the bit is zero, the terminal is requested to acknowledge the block immediately (by sending an ACK block in reply). The terminal sends the ACK block immediately, whether or not it has a message to pack into that block. (The ACK block contains only the four block control bytes.)

If, however, the host sets Bit 7 to one, then the terminal does not acknowledge the block immediately. Instead, it waits until it has a block full of data to send, or until it encounters an <EOM-char> or <EOM-indicator> in the data it has to send to the host.

When sending a block to the terminal, the host should set Bit 7 to zero, except when it expects a response message from the terminal. If a response message is expected, the host should set Bit 7 to one.

<Control-Byte-2>

All the bits of <control-byte-2> are reserved; they are always zero.

Table 11-2

MEANINGS OF LOW-ORDER BITS IN <CONTROL-BYTE-1>

Bit 2	Bit 1	Meaning
0	0	This is an "even" block, and no attempt is being made to remove the terminal from block mode.
0	1	This is an "odd" block, and no attempt is being made to remove the terminal from block mode.
1	0	In a block sent from the host to the terminal, these two bits comprise a command to the terminal: "Exit from block mode, but remain armed for block mode. Before exiting block mode, however, acknowledge this command by sending an 'ACK' block to the host." In a block sent from the terminal to the host, these two bits mean, "This is an 'ACK' block acknowledging receipt of a command to exit block mode. The terminal is now leaving block mode, but will remain armed for block mode."
1	1	In a block sent from the host to the terminal, these two bits comprise a command to the terminal: "Exit from block mode, but remain armed for block mode. Exit block mode immediately; do not send an 'ACK' block to the host." In a block sent from the terminal to the host, this combination of bits is not allowed.

<Control-Byte-3> and <Control-Byte-4>

The last two control bytes carry a "check code" by which the receiving device (the terminal or the host computer) can verify that it has received the block with no errors. The check code is derived from all the unpacked data bytes which precede it: all the 7- or 8-bit bytes of meaningful data, plus the first two control bytes. The process is as follows:

1. Two "checksum bytes" — called H and L for this explanation — are both set equal to MaxByte. Here, MaxByte is 127 (for 7-bit bytes) or 255 (for 8-bit bytes).

2. Each byte in the preceding unpacked data is regarded as a binary numeral and added to L. The sum is computed as with "modulo 7 (or modulo 8) end-around-carry." That is, for a 7-bit "unpacked byte size," whenever the sum exceeds the maximum 7-bit numeral (127), the "carry" bit is omitted and one is added to the least-significant bit of the sum. Likewise, if the unpacked byte size is 8-bits, then whenever the sum exceeds 255, the carry bit is omitted and one is added to the least-significant bit of the sum.

   This process is equivalent to the following algorithm, in which MaxByte = 127 (for 7-bit bytes) or 255 (for 8-bit bytes):

   ```
   BEGIN
     L := L + Byte;
     IF (L > MaxByte) THEN L := L — MaxByte
   END
   ```

3. As each byte is added to L, the new value of L is added to H. The same "end-around-carry" method is used:

   ```
   BEGIN
     H := H + L;
     IF (H > MaxByte) THEN H := H — MaxByte
   END
   ```

4. When Steps 2 and 3 have been performed for each of the unpacked bytes preceding the check code bytes, then the two check code bytes are computed as follows:

   ```
   BEGIN
     C1 := MaxByte — H — L;
     IF (C1 < 1) THEN C1 := C1 + MaxByte;
     ControlByte3 := C1;
     ControlByte4 := H
   END
   ```

Packing the Control Bytes into the Block. When all four control bytes have been computed, they are packed into the block along with any other unpacked data bytes; see the description of the <set-block-packing> command for details.

Checking a Received Block. When a block is received and unpacked, the H and L checksum bytes are computed as described previously. As each byte is unpacked, the "unpacked byte" is added to L (with end-around carry), and L is added to H (with end-around carry). This is done on all bytes as they are unpacked, including all four control bytes. When the <block-end-char> is detected, H and L should both equal MaxByte. That is, if the unpacked bytes are 7-bit bytes, then H = L = 127; if they are 8-bit bytes, H = L = 255. If this is not the case, then a data transmission error has occurred. (In that case, the terminal or the host receiving the block would retransmit the last block it had sent.)

RETRANSMITTING BAD BLOCKS

ACK Blocks and NAK Blocks

Recall, from the description of the block control bytes, that the least significant bit of <control-byte-1> serves as a modulo two "odd/even" counter. This counter is used to identify a block as an "ACK block" or a "NAK block."

Host to Terminal. When the host sends a block to the terminal, the terminal acknowledges correct reception of that block by sending an "ACK block" back to the host. This is a block with the same block count bit as the block which was correctly received.

If, however, the terminal detects a checksum error in the block, then it retransmits to the host the last previous block that it sent the host. The retransmitted block has a block count bit which is different from that in the block just (incorrectly) received. The fact that the block count bit is different identifies that block to the host as a "NAK block."

If the host receives a NAK block, it retransmits the block it just sent to the terminal. If the host receives an ACK block, then the next block it sends will contain new data.

Terminal to Host. Likewise, when the terminal sends a block to the host, the host acknowledges correct reception of the block by sending an "ACK block" back to the terminal. In this case, however, an ACK block is defined as a block with the *opposite* block count bit.

If the host, by means of the checksum bytes, detects an error in the block, it retransmits to the terminal the previous block it had sent the terminal. That retransmitted block has a block count bit which is the same as the block count bit in the block which the host just (incorrectly) received. The fact that the block count bit is the same identifies that block to the terminal as a "NAK block."

If the terminal receives a NAK block, it retransmits the block it just sent. If, on the other hand, the terminal receives an ACK block, then the next block it sends to the host can contain new data.

Normal, Error-Free Transmission

Figure 11-1 shows normal transmission of data from the host to the terminal. If the terminal has been armed for block mode, then it enters block mode on receiving the header for the first block sent from the host.

Host's Point of View. From the host's point of view, each "block transaction" consists of sending a block of data to the terminal and receiving an ACK block in return. (Here, an ACK block is a block with the same even/odd count as the block just sent.) Figure 11-1 shows three such transactions. In the first block transaction, the host sends an odd-numbered block to the terminal and receives an odd-numbered block in return. In the second transaction, the host sends an even block and receives an even block in return. In the third transaction, the host sends an odd block and receives an odd block in return.

Terminal's Point of View. From the terminal's point of view, each "block transaction" consists of sending a block to the host and receiving an ACK block in return. (Here, an ACK block is a block with the opposite odd/even count as the block just sent.) From the terminal's point of view, Figure 11-1 shows two complete block transactions, plus a third transaction which is not yet complete.

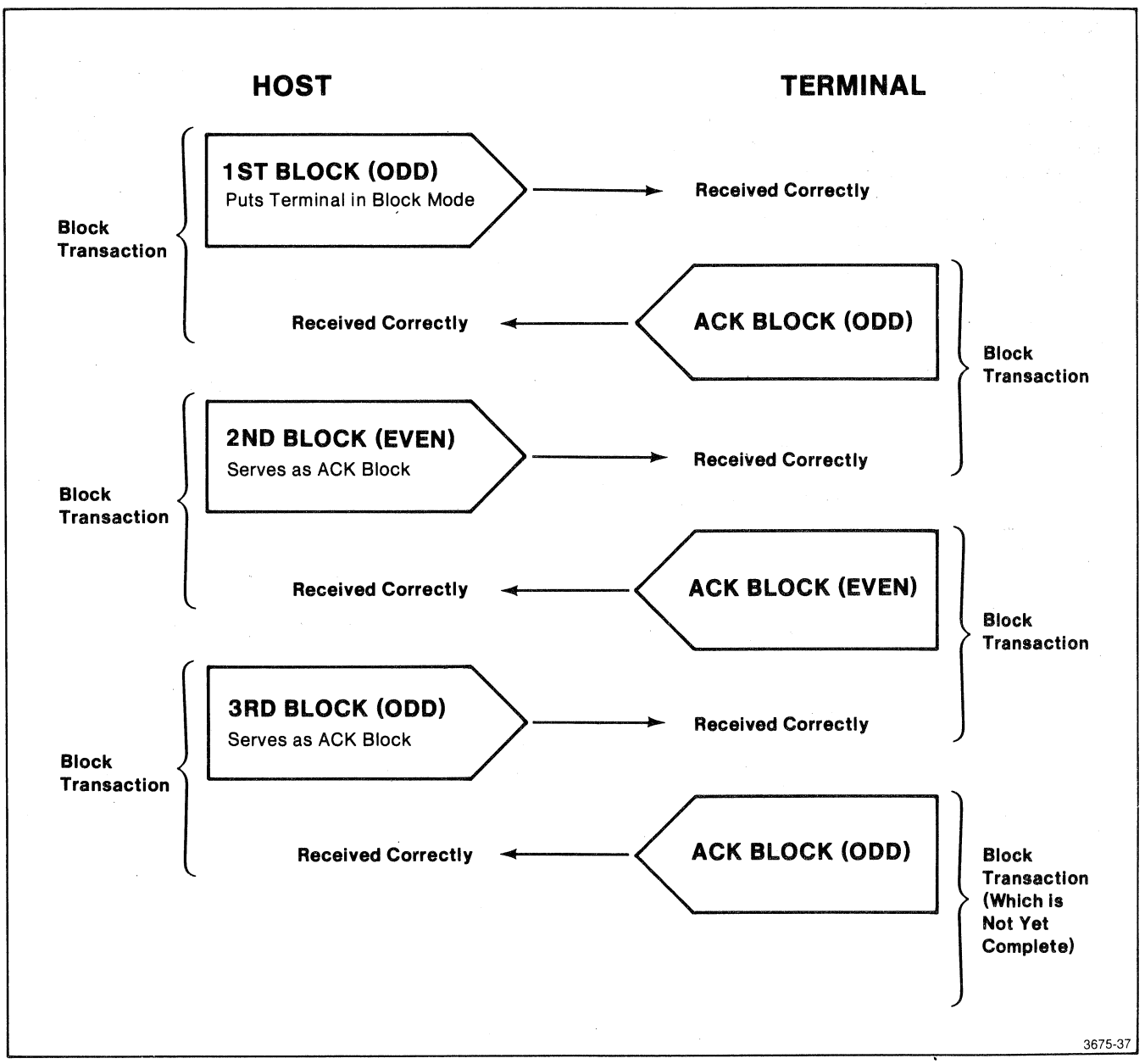


Figure 11-1. Error-Free Transmission from Host to Terminal.

Effect of Occasional Errors

Figure 11-2, a continuation of Figure 11-1, shows the effect of an occasional noise burst on the data communications line. When the host sends the fourth block to the terminal, noise on the communications line causes that block to be received incorrectly by the terminal. The terminal detects a checksum error and retransmits the previous block. The host interprets that block as a NAK block, and retransmits the fourth block. This time, the terminal receives the block correctly.

Effect of Multiple Errors

Figure 11-3, which continues from Figure 11-2, shows the effect of multiple data communications errors. Whenever the checksums do not agree, the host and terminal keep retransmitting the previous blocks sent. Provided the noise on the data communications line is not continuous, eventually the data will be transferred successfully.

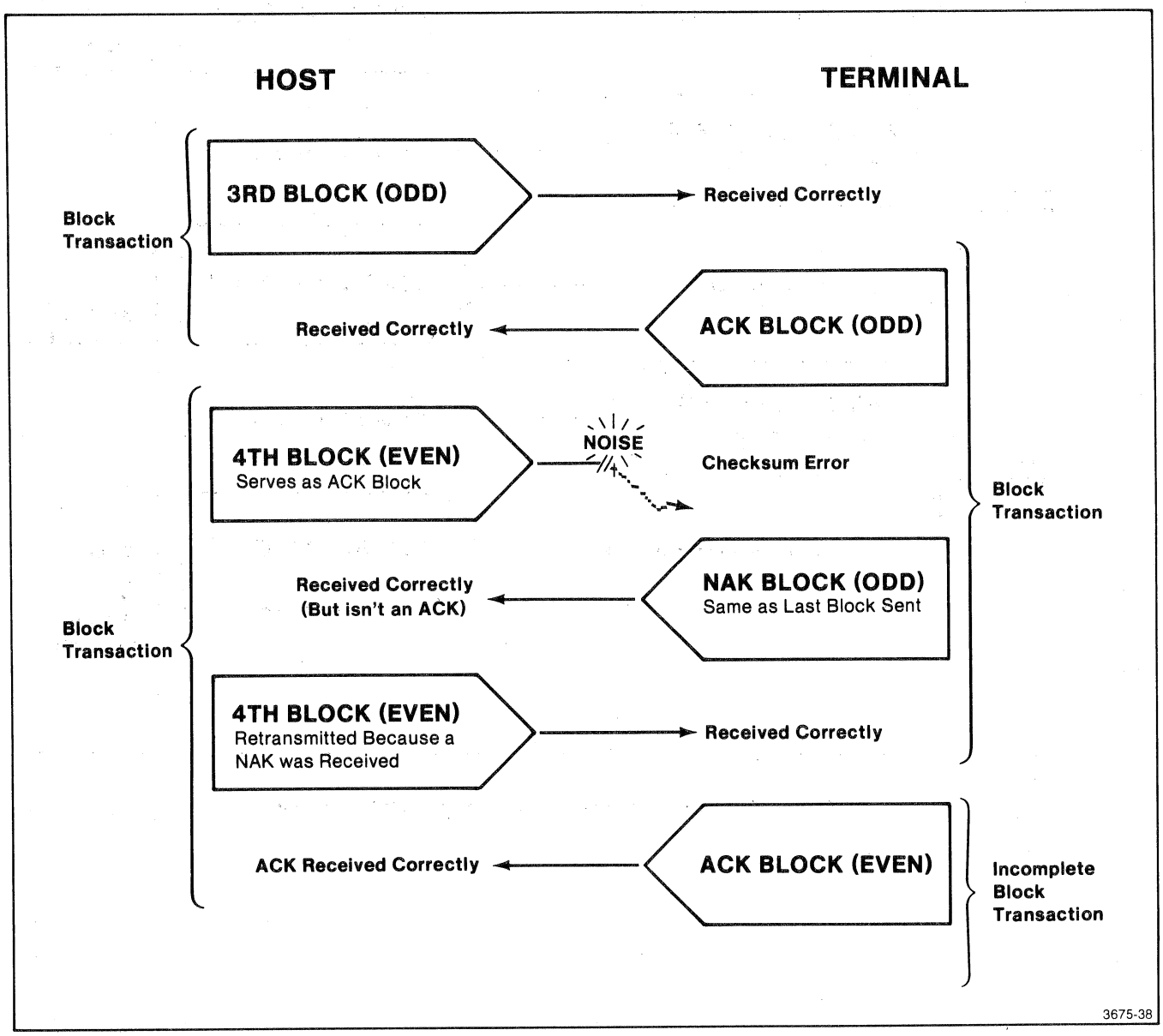


Figure 11-2. Effect of Occasional Errors in Block Mode Transmission.

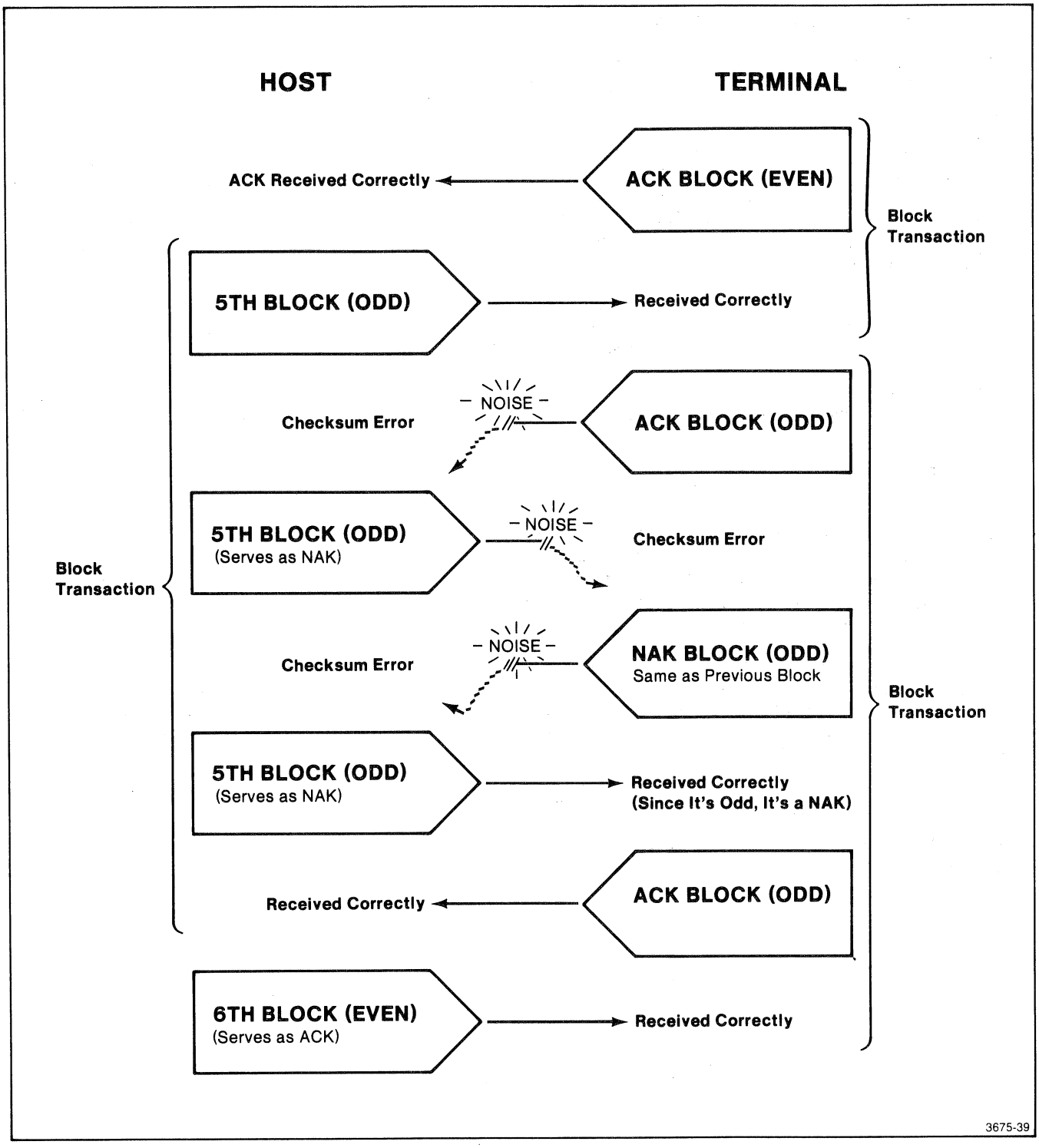


Figure 11-3. Effect of Multiple Errors in Block Mode Transmission.

<Set-Block-Timeout> Command

After receiving a block from the host, the terminal ignores further characters coming from the host until it receives the header string that starts a new block.

But what happens if the header string itself is garbled by noise on the line? In that case, the terminal would not detect the start of a new block and so could not send a NAK block in response to the (garbled) block. The system would "hang," with the terminal waiting for a block that never comes.

All is not lost; there is provision for that situation, too. The terminal has a "block timeout" feature, which can be set with the <set-block-timeout> command.

Before arming the terminal for block mode, the host issues a <set-block-timeout: 10> command, as follows:

<set-block-timeout: 10> = (ESC)(O)(T)<int: 10>
= (ESC)(O)(T)(:)

This sets the block timeout to ten seconds. With this setting, if the terminal sends a block to the host and does not receive any block back within ten seconds, then the terminal interprets the *absence* of a reply as a NAK, and re-transmits the block just sent.

You can disable the timeout feature with a <set-block-timeout: 0> command. If you use the timeout feature, you should set the timeout period to a duration which is longer than your host's maximum expected response time.

The operator can also issue the <set-block-timeout> command, using the SETUP mode command, BTIMEOUT. See the 4113 Operator's Manual for details.

PROGRAMMING CONSIDERATIONS

The following hints should help you when programming the host computer for block mode.

- The first block sent, which puts the terminal in block mode, should be an odd-numbered block. (If the terminal receives this block with a checksum error, the NAK block it sends in reply is even-numbered. For the host to interpret the NAK correctly, it should be set to interpret even-numbered blocks as NAKs. That is the case if the host has just sent an odd-numbered block to the terminal.)
- All block mode parameters should be set before issuing the <arm-for-block-mode> command. (The commands to set these parameters are invalid if the terminal is armed for block mode.)
- When the host expects the operator to type on the keyboard (or the terminal to send a report message), it should send a block to the terminal with the end-of-message bit set to one. When the host is sending commands to the terminal and does not expect a reply (other than ACK blocks), it should set the end-of-message bit to zero in blocks sent to the terminal.

  If the terminal receives a block in which the end-of-message bit is zero, it acknowledges that block immediately: it sends an ACK block which contains only the four block control bytes. If, however, the end-of-message bit is one, then the terminal does not immediately acknowledge the block. Instead, it waits until it has data (besides the block control bytes) to pack into the ACK block.

See Appendix C for examples of FORTRAN block mode communications drivers.

Section 12

PERIPHERAL DATA TRANSFERS

INTRODUCTION

This section describes data transfers between the terminal, the host computer, and optional peripheral devices such as a flexible disk drive or a plotter.

This section includes the following major topics:

- **Overview of Commands.** This introduces the abbreviations for the different peripheral devices, and shows how these abbreviations are used in one data transfer command, the <copy> command. The commands described later in the section are then listed, with brief descriptions of each command.
- **Using the Disk Drive.** The commands associated with Option 42 and 43 (the disk drives) are given here, with examples of their use.
- **Initializing the RS-232 Peripheral Ports.** This introduces commands to set parameters for the Option 10 Three Port Peripheral Interface.
- **Using a Printer.** This gives examples of commands to configure an RS-232 peripheral port for use with an external printer, and of commands to print information on that printer.
- **Using a Plotter.** This gives examples of commands to configure a peripheral port for use with a TEKTRONIX 4662 or 4663 Interactive Digital Plotter, and of commands to send graphic information to that plotter.
- **Using Other RS-232 Devices.**

OVERVIEW OF COMMANDS

COMMAND FORMAT

The terminal's data transfer commands can be typed by the operator in SETUP mode or sent to the terminal as escape sequences from the host. The escape-sequence versions of the commands closely resemble the SETUP mode versions.

For instance, consider the COPY command. In SETUP mode, the operator can type the following command:

COPY HO: TO F0:FILE1

This causes the terminal to copy data coming from the host computer ("HO:") to a file named FILE1 on disk drive zero ("F0:FILE1"). Disk drive zero ("F0:") is located just above the keyboard, on the right. Disk drive one ("F1:") is to the left of drive one. Drive zero is provided with Option 42, while both drives are provided with Option 43.

Note that there are three parameters in this COPY command. The first parameter ("HO:") denotes the source of the data. The second parameter ("TO") is included to remind the operator of the direction of data transfer. The third parameter ("F0:FILE1") names the destination for the data transfer.

The corresponding escape-sequence <copy> command has this syntax:

<copy> = (ESC)(J)(C)
<string: *source-specifier*>
<string: empty-string or "TO">
<string: *destination-specifier*> .

Note that there are three parameters, all of type <string>, corresponding to the three parameters in the SETUP mode COPY command. The second parameter may be the empty <string>, or it may be the <string> holding only the letters T and O.

Since the host computer must issue commands to the terminal as escape sequences, it would send the "COPY HO: TO F0:FILE1.DAT" command as follows:

<copy: from host to F0:FILE1.DAT>

= (ESC)(J)(C)
<string: "HO:">
<string: empty-string>
<string: "F0:FILE1.DAT">

= (ESC)(J)(C)
(3)(H)(O)(:)
(0)
(<)(F)(0)(:)(F)(I)(L)(E)(1)(.)(D)(A)(T)

= (ESC)(J)(C)(3)(H)(O)(:)(0)(<)(F)(0)(:)
(F)(I)(L)(E)(1)(.)(D)(A)(T) .

DEVICE SPECIFIERS

In data transfer commands, the source and destination devices are represented with three-character device specifiers. Examples of these are "HO:" and "F0:", used in the preceding example to represent "the host computer" and "flexible disk drive number zero." Table 12-1 lists the valid device specifiers.

Table 12-1

DEVICE SPECIFIERS

Device Specifier	Meaning
HO:	The host computer.
FO: F1:	Disk drives. (Requires either Option 42 or 43.)
PO: P1: P2:	RS-232 peripheral ports. (Requires Option 10.)
HC:	Color Hard Copier Interface. (Requires Option 09.)
SC:	Pseudodevice corresponding to screen contents. (Requires Option 09.)

FILE NAMES

If the source or destination for a data transfer is a disk file, that file is represented by a <string> parameter such as "F0:FILENAME" or just "FILENAME". Here, "F0:" is the device specifier for the disk drive. "FILENAME" is the name of the file on that drive; it is a sequence of up to nine letters, digits, and periods (decimal points). The terminal does not distinguish between uppercase and lowercase letters; "f0:file1" refers to the same disk file as "F0:FILE1". The disk drive specifier ("F0:") may be omitted, in which case drive zero is assumed. That is, "FILE2.DAT" refers to the same file as "F0:FILE2.DAT".

DATA-TRANSFER COMMANDS

Table 12-2 lists the data-transfer commands. These commands are described later in this section. For more detailed information, see the command descriptions in the 4110 Series Command Reference Manual.

Table 12-2

DATA-TRANSFER COMMANDS

Command Name	Description
<Copy>	Copies data from a "source" to a "destination."
<Spool>	Similar to <copy>, but the terminal can be used for other functions while the <spool> operation proceeds.
<Port-Copy>	Establishes a bidirectional data path to a peripheral port.
<Save>	Encodes a graphic segment, a macro definition, or contents of the pixel viewport as a series of escape-sequence commands, and sends those commands to the destination device.
<Load>	Takes commands from the specified source device and executes them. For instance, after a segment has been <save>d to a disk file, it can be re-created by <load>ing that disk file.
<Directory>	Compiles a directory of the files on a diskette and sends that directory to the specified destination. If no destination is specified, the directory is displayed for the operator to see.
<Plot>	Takes the current settings for the current view's window and viewport, encodes them as <set-window> and <set-viewport> escape-sequence commands, and sends those commands to the specified destination. Likewise, takes all segments visible in the current view and sends them as escape-sequence commands to the specified destination. If the destination is an RS-232 peripheral port to which a TEKTRONIX 4662 or 4663 plotter is attached, and a <port-assign> command has assigned the appropriate plotter device driver to that port, then the <plot> command causes the contents of the current viewport to be drawn on the plotter.

FORMATTING AND PARAMETER-SETTING COMMANDS

Table 12-3 lists commands which prepare the terminal (or, in the case of the <format-volume> command, the flexible diskette) for subsequent data-transfer commands.

Table 12-3

FORMATTING AND PARAMETER-SETTING COMMANDS

Command Name	Description
<Format-Volume>	Writes formatting information on a flexible diskette to prepare the diskette to hold data files.
<Port-Assign>	Assigns a "device-driver protocol" to an RS-232 peripheral port. It is by the <port-assign> command that you inform the terminal what kind of device is attached to the specified port.
<Set-Port-Baud-Rate> <Set-Port-EOF-String> <Set-Port-Flagging-Mode> <Set-Port-Parity> <Set-Port-Stop-Bits> <Set-Port-EOL-String>	These commands set baud rate, parity, and other settings for RS-232 peripheral ports.

COMMANDS TO REPORT PERIPHERAL STATUS

Table 12-4 lists commands by which the host computer can obtain status information about peripheral devices.

Table 12-4

COMMANDS TO REPORT PERIPHERAL STATUS

Command Name	Description
<Report-Device-Status>	Causes the terminal to send a <device-status-report> to the host computer. The report contains a status integer in which is encoded such information as: whether the device is present, whether the device is busy, etc.
<Report-Port-Status>	Causes the terminal to compose a <port-status-report> for the specified peripheral port, and to send that <port-status-report> to the host computer. The report message contains information on such settings as baud rate, parity, stop bits, etc.

USING THE DISK DRIVES (OPTIONS 42 AND 43)

The Option 42 and 43 Disk Drives provide local mass storage of graphic segments, macro definitions, and other sequences of commands to the terminal.

<FORMAT-VOLUME> COMMAND

Before a diskette can store data, it must be formatted. Normally, this is done by the operator, using the SETUP mode FORMAT command; see the 4113 Operator's Manual for details. However, this can also be done from the host computer, using the escape-sequence command, <format-volume>:

<format-volume> = (ESC)(J)(F)
<string: *volume-specifier*>
<int: *number-of-files*> .

The volume-specifier contains a volume name which will be displayed in response to <directory> commands for that diskette, e.g. "VOLUME1". This name may contain up to nine letters and digits. If you wish, you can precede the volume name with the characters "F0:"or "F1:" to designate disk drive zero or disk drive one. If you omit the drive specifier, "F0:" is assumed.

The <int> parameter specifies the number of files permitted in the diskette being formatted. The number actually formatted is a multiple of sixteen; the smallest multiple of sixteen which is at least as large as the number specified in this parameter.

Of course, the diskette must not be protected with a write-protect switch or write-protect notch. (See the 4113 Operator's Manual for details.)

<COPY> FROM HOST TO DISK FILE

A host program can use the <copy> command to create a data file on a diskette in the terminal's disk drive. The process is as follows:

1. Issue a <copy> command to the terminal. The first parameter in this command is <string: "HO:">, specifying the host as the source of the data to be transferred. The second parameter is the empty string or <string: "TO">. The third parameter is a <string> specifying the file to which the data is to be copied. For example:

   <copy: "HO:", empty-string, "F0:DATA">

   = (ESC)(J)(C)
   <string: "HO:">
   <string: empty-string>
   <string: "F0:DATA">

   = (ESC)(J)(C)(3)(H)(O)(:)(0)(7)(F)(0)(:)
   (D)(A)(T)(A) .

2. Send to the terminal the data which is to be included in the file. (This could, for instance, consist of escape-sequence commands which are later to be <load>ed from the disk file.)
3. Send to the terminal the current <EOF-string>, as set by the most recent <set-EOF-string> command. This is a sequence of up to ten characters which signals the end of the data transfer. (The <set-EOF-string> command is described later in this section. Like all commands, its definitive description is to be found in the 4110 Series Command Reference manual.)

<EOF-String>s and the <Set-EOF-String> Command

When the <copy> command transfers data to or from the host, the end of the data transfer is marked with an <EOF-string> (end-of-file string). This is a sequence of up to ten ASCII characters, set by the <set-EOF-string> command. That command has the following syntax:

<set-EOF-string> = (ESC)(N)(E)
<int-array: *ASCII-decimal-equivalents*> .

The numbers in the <int-array> parameter are the numeric equivalents of the ASCII characters which make up the end-of-file string.

An Example

You can create a disk file which the operator can later <load> into the terminal so as to initialize the terminal for a certain task. For instance, the file might contain <define-macro> commands to program certain of the terminal's function keys. Or, it might contain commands to set the terminal's baud rate, parity, etc., for communicating with a certain host computer. To do this, send a sequence of commands from the host such as the following:

```
<set-EOF-string: (47,42,42,47)>
<copy: "HO:", "TO", "F0:COMMANDS">
    <define-macro>
    <define-macro>
    <define-macro>
    .
    .
    .
    <set-baud-rates>
    <set-parity>
    .
    .
    .
(/)(*)(*)(/)
<set-EOF-string: empty-array>
```

Here, the first <set-EOF-string> command specifies "/**/" as the end-of-file string. (The numbers 47 and 42 are the decimal equivalents of the characters (/) and (*), used in the <EOF-string>.)

The <copy> command starts the data transfer. The host is the source of data, and the file named COMMANDS on disk drive zero is the destination.

The following commands (<define-macro>, <set-baud-rates>, etc.) are part of the data being transferred. The terminal does not execute these commands, but just copies them into the destination file (the file named COMMANDS, on disk drive zero).

The characters (/)(*)(*)(/) are the <EOF-string>; they mark the end of the data transfer. When the terminal encounters these characters, it closes the disk file F0:COMMANDS and ceases to execute the <copy> command.

The final <set-EOF-string> command sets the <EOF-string> to the empty string. This command is as follows:

<set-EOF-string: empty-array>

= (ESC)(N)(E)
<int-array: empty>

= (ESC)(N)(E)(0) .

You may be wondering why this last command is necessary. It is needed because the terminal suppresses <EOF-string>s in the data it reads from the host. After the file transfer, it is possible that the host might have occasion to send the character sequence (/)(*)(*)(/), perhaps as <xy> coordinates in graphic data, or as <int> parameters for some command. Should that occur, the terminal would still recognize those characters as the current <EOF-string>, and would remove them from the data it receives. This could cause undesired results. The problem can be avoided by (a) choosing an <EOF-string> which is unlikely to occur in data sent to the terminal, or (b) by setting the <EOF-string> to the empty string except when it is needed for a data transfer. The latter approach was used in this example.

<COPY> FROM DISK FILE TO HOST

The <EOF-string> plays the same role in a <copy> to the host computer as it does in a <copy> from the host to the terminal. This time, however, it is the terminal which issues the <EOF-string>, and the host which must recognize that <EOF-string> as marking the end of the data transfer.

<DIRECTORY> COMMAND

The <directory> command prepares a disk directory for the specified disk drive, and sends that directory to a specified destination. (If no destination is specified, the directory information is sent to the terminal's display.) The escape-sequence command has the following syntax:

<directory> = (ESC)(J)(D)
<string: *disk-drive-specifier*>
<string: empty-string or "TO">
<string: *destination-specifier*> .

The SETUP mode version of this command is described in the operator's manual. The following examples show how it might be used:

DIRECTORY F0: Displays a directory of the files on the diskette in disk drive zero. (Since the second and third parameters are omitted, the directory is sent to the terminal's display: to the dialog area if the dialog area is enabled, otherwise to the current graphic beam position.)

DIRECTORY F0: TO F0:DIRECTORY
Composes a directory of the files on the diskette in disk drive zero, and records that directory in the file named DIRECTORY on that disk drive.

DIR F0: TO P1: Composes a directory of the files on disk drive zero, and sends that directory to RS-232 peripheral port number one. If a printer is attached to that port, then the file directory is printed. If a plotter is attached to that port, then the plotter "prints" the file directory.

DIR F0: TO HO: Sends to the host computer a directory of the files on disk drive zero. At the end of the data transfer, the terminal sends the current <EOF-string>, as set by the most recent <set-EOF-string> command.

The host computer can issue <directory> commands by using the escape-sequence syntax just described. For instance, the host's equivalent of DIR F0: TO P0: would be the following:

<directory: "F0:", empty-string, "P0:">

= (ESC)(J)(D)
<string: "F0:">
<string: empty-string>
<string: "P0:">

= (ESC)(J)(D)
(3)(F)(0)(:)
(0)
(3)(P)(0)(:)

= (ESC)(J)(D)(3)(F)(0)(:)(0)(3)(P)(0)(:) .

Figure 12-1 shows the result of a typical <directory> command, as displayed on the terminal's screen.

```
-NAME          BLOCKS BYTES PROTECT
F0:K7KK          2002         NO
   KITTY            9     6   NO
   DSTAR           46    56   NO
   BUMP            34    84   NO
   DRAGON          15   107   NO
   GRINCH           9   154   NO
   OSU1             7    22   NO
   LASERGUN        28   109   NO
   MAZE             6   162   NO
   MAGIC            5    18   NO
   MICKEY           4   121   NO
   PARAB            4   147   NO
   R2D2            28   103   NO
   SNOOPY           2   205   NO
   STAR             5   213   NO
   SYMBOL          10   104   NO
   WIZARD           9    75   NO
   SEGMENT1        10   153   NO
   PATTERNS         4   206   NO
   PATTERNS2       10    27   NO
-
-ENTRIES USED:    19
-ENTRIES FREE:   349
-BLOCKS USED:    271
-BLOCKS FREE:   1731
-LARGEST FREE:  1715
```

3892-4

Figure 12-1. <Directory> Command Report Format.

<Directory> to a Printer

If the terminal is equipped with Option 10 (Three Port Peripheral Interface), then you can send <directory> reports to a printer attached to one of the RS-232 peripheral ports. For this to work properly, the following conditions must be met:

- The peripheral port must have a "4643" device protocol assigned to it. (See the description of the <port-assign> command, later in this section.)
- The peripheral port's end-of-line string must be set to (CR)(LF). (See the <set-port-EOL-string> command description for details.) This is necessary in order for the terminal to send (CR)(LF), rather than just (CR), at the end of each line in the <directory> message.
- The peripheral port baud rate, parity, etc., must be set to match the corresponding values for the printer connected to that port.

Once these conditions are met, the operator and host computer can issue a <directory> command with the printer port as the specified destination device. For instance, if a printer is attached to peripheral port number one, the operator might type the following SETUP mode command:

DIRECTORY F0: TO P1:

This causes a directory of the files on disk drive zero ("F0:") to be printed on a printer attached to peripheral port number one ("P1:"). The corresponding escape-sequence command from the host computer would be:

<directory: "F0:", empty-string, "P1:">

= (ESC)(J)(D)
  <string: "F0:">
  <string: empty-string>
  <string: "P1:">

= (ESC)(J)(D)(3)(F)(0)(:)(0)(3)(P)(1)(:) .

<Directory> to a Plotter

If the terminal is equipped with Option 10 (Three Port Peripheral Interface), then you can send <directory> reports to a TEKTRONIX 4662 or 4663 Interactive Digital Plotter. For this to work properly, the following conditions must be met:

- The plotter must be set for "CR implies CR-LF" mode. This is because each line of the directory report ends with a single (CR) character — *not* (CR)(LF).
- Other plotter parameters must be set as follows:

  Attention character (ESC)
  Address character (A)
- The peripheral port must have a "4662/NT" or "4663/NT" protocol assigned to it. (See the description of the <port-assign> command, later in this section.) The port baud rate, stop bits, etc., must be set to agree with the corresponding plotter settings.
- The operator must position the plotter pen (with the joystick) to the upper left corner of the plotter's viewport.

Once the plotter and the peripheral port have been set up this way, the operator or the host computer can issue a <directory> command and let the plotter draw the resulting file directory. For instance, if the plotter is at peripheral port zero, the operator types the following SETUP mode command:

DIRECTORY F0: TO P0:

To produce the same result, the host computer would issue the corresponding escape-sequence command:

<directory: "F0:", empty-string, "P0:">

= (ESC)(J)(D)
  <string: "F0:">
  <string: empty-string>
  <string: "P0:">

= (ESC)(J)(D)(3)(F)(0)(:)(0)(3)(P)(0)(:) .

<Directory> to the Host Computer

The host can cause the terminal to send it a file directory, as follows:

<directory: "F0:", empty-string, "HO:">

= (ESC)(J)(D)
  <string: "F0:">
  <string: empty-string>
  <string: "HO:">

= (ESC)(J)(D)(3)(F)(0)(:)(0)(3)(H)(O)(:) .

As with the <copy> command, the end of the data transfer is marked with an <EOF-string>. (In block mode, the end of the data is marked with the end-of-message bit in the block control bytes. See Section 11 for details.)

Each line of the message sent to the host ends with a single (CR) character — *not* the character sequence (CR)(LF).

<RENAME-FILE> COMMAND

The <rename-file> command lets you assign a new name to a disk file. For instance, in SETUP mode, the operator can type:

RENAME FILE1 TO FILE2

This takes the file F0:FILE1 (the file named FILE1 on disk drive zero) and renames it F0:FILE2.

The escape-sequence syntax for this command is as follows:

<rename-file> = (ESC)(J)(R)
<string: *old-file-specifier*>
<string: empty-string or "TO">
<string: *new-file-specifier*> .

For more details, see the description of the <rename-file> command in the 4110 Series Command Reference Manual.

<DELETE-FILE> COMMAND

The <delete-file> command lets you delete a disk file. For instance, in SETUP mode, the operator can type:

DELETE F0:FILE1

This deletes the file named FILE1 from the diskette in disk drive zero. The escape-sequence syntax for this command is as follows:

<delete-file> = (ESC)(J)(K)
<string: *file-specifier*> .

For instance, to delete the file F0:ABCDEF, the host program can issue the following command:

<delete-file: "F0:ABCDEF">

= (ESC)(J)(K)<string: "F0:ABCDEF">

= (ES)(J)(K)(9)(A)(B)(C)(D)(E)(F) .

For more details, see the description of this command in the 4110 Series Command Reference Manual.

<LOAD> COMMAND

The <load> command causes a disk file to be executed by the terminal, just as if the commands (or other data) in the file were sent to the terminal by the host computer.

Suppose, for instance, that the command file F0:INITIALIZ has been created by <copy>ing data from the host computer. In the file are escape-sequence commands for the terminal. To cause these commands to be executed, the operator puts the terminal in SETUP mode and types:

LOAD F0:INITIALIZ

In response to this command, the terminal reads the data in the file, treating that data in the same way as it treats data coming from the host computer. The terminal executes any escape-sequence commands which it finds in the file.

The escape-sequence version of this command is as follows:

<load> = (ESC)(J)(L)
<string: *file-specifier*> .

For more details, see the description of this command in the 4110 Series Command Reference Manual.

<SAVE> COMMAND

The <save> command lets you save macro definitions, graphic segments, or the contents of the pixel viewport as escape-sequence commands in a disk file. Later, the file can be <load>ed to restore to the terminal the information which was <save>d.

The operator can type this command in SETUP mode, or it may be sent as an escape-sequence command from the host computer. The following examples show how an operator might use this command:

SAVE SEG 1 TO F0:SEGMENT1
: Saves graphic segment number one as a series of escape-sequence commands on disk drive zero, in the file named SEGMENT1.

SAVE MAC 128 TO MACRO.65
: Saves macro definition number 128 (the macro which may be invoked with function key F1) as a series of escape-sequence commands in the file named MACRO.65 on disk drive zero.

SAVE SEG −1 TO SEGMENTS
: Saves the definitions of *all* graphic segments in a file named SEGMENTS on disk drive zero.

SAVE RAS 640 TO FILE1
: Saves 640 pixels from the current pixel viewport as <raster-write> commands in the file named FILE1 on disk drive zero.

SAVE RUN −1 TO FILE1
: Saves *all* pixels in the pixel viewport, writing them as <run-length-write> commands in the disk file named FILE1.

The escape-sequence syntax for this command is as follows:

<save> = (ESC)(J)(V)
<string: *code-for-thing-to-be-saved*>
<int: *item-number-or-count*>
<string: empty-string or "TO">
<string: *destination-specifier*>

<string: *code-for-thing-to-be-saved*>
= <string: "SEG">
or <string: "MAC">
or <string: "RAS">
or <string: "RUN">

For instance, the host computer could define a segment, save it on a disk file, and delete the segment definition by issuing commands such as the following:

```
<delete-segment: 5>
<begin-segment: 5>
    <enter-vector-mode>
        <xy>
        <xy>
        .
        .
        .
    <enter-vector-mode>
        <xy>
        <xy>
        .
        .
        .
    .
    .
    .
<end-segment>
<save: "SEG", 5, "TO", "F0:SEGMENT5">
<delete-segment: 5>
```

Later, the segment definition can be recalled from the disk with a <load> command:

<load: "F0:SEGMENT5">

The <save> command can have other destinations than a disk file. For instance, the operator can type the following in SETUP mode:

SAVE SEG 3 TO P0:

If a plotter is connected at port zero, this command would cause segment 3 to be drawn by that plotter. (For this to work properly, port zero must have the appropriate plotter protocol assigned to it, and the plotter itself must be set up properly. For the segment to be mapped onto the plotter using the current window-viewport transform, the <plot> command, rather that <save>, should be used. These details are described later in this section.)

For other information about the <save> command, see the command description in the 4110 Series Command Reference Manual.

INITIALIZING THE RS-232 PERIPHERAL PORTS (OPTION 10)

The Option 10 Three Port Peripheral Interface provides three RS-232 peripheral ports to which may be attached such peripheral devices as a TEKTRONIX 4641 Printer, or TEKTRONIX 4662 or 4663 Interactive Digital Plotters.

<PORT-ASSIGN> COMMAND

The <port-assign> command assigns a "device protocol" to a particular RS-232 peripheral port.

The SETUP mode name for this command is PASSIGN. The following examples show how the operator uses the PASSIGN command when attaching peripherals to the terminal:

PASSIGN P0: PPORT Assigns a general-purpose RS-232 protocol to RS-232 peripheral port zero. The PPORT protocol makes no assumptions about the nature of a device which may be attached to this port.

PASSIGN P1: 4643 Assigns the 4643 protocol to peripheral port one. This is similar to the PPORT protocol. However, any (CR) characters occurring in data sent to this port will be replaced with the port's current end-of-line string, as set by the <set-port-EOL-string> command.

If you want to print data (such as a <directory> report) in which lines end with (CR) rather than (CR)(LF), set the port's <EOL-string> to (CR)(LF). (Use the <set-port-EOL-string> command, described later in this section.) That way, each (CR) in the file is replaced with (CR)(LF), and the printer displays each line of the file on a new line of the output paper. This is especially convenient for doing <directory> commands with the printer as a destination.

If the data to be printed already includes (CR)(LF) sequences at the end of each line, then you would set the port's end-of-line string to just (CR).

PASSIGN P0: 4662 Assigns a "4662" device-driver protocol to peripheral port zero. With this protocol assigned, the terminal assumes that a TEKTRONIX 4662 Interactive Digital Plotter is connected to the peripheral port.

Any data being sent to port zero is sent using the 4662's block mode communications protocol. Carriage returns and line feeds are translated into "move" commands for the plotter, and 4110-style escape-sequence commands are translated into equivalent plotter-language commcommands.[1]

PASSIGN P0: 4662/NT Similar to the above; the terminal assumes that the device attached to this port is a 4662 plotter, and uses the plotter's block mode protocol for data sent out this port. However, NO TRANSLATION of carriage returns, line feeds, or 4110-series escape-sequence commands is performed.[1]

PASSIGN P0: 4662/MP Informs the terminal that a 4662 plotter with the MULTIPLE PEN option is connected to port zero. As with the "4662" protocol, the plotter's block mode is used, and 4110-series commands are translated into corresponding commands for the plotter.

PASSIGN P2: 4663 Assigns a "4663" protocol to peripheral port two. This informs the terminal that a TEKTRONIX 4663 Interactive Digital Plotter is connected to the port.[1]

As with the "4662" protocol, the plotter's block mode is used. As with the "4662" protocol, carriage returns, line feeds, and 4110-series escape-sequence commands are translated into corresponding plotter-language commands.[1]

[1] See NOTE under "Using a Plotter."

PASSIGN P2: 4663/NT
Similar to "4663" protocol, except that NO TRANSLATION to plotter language is performed.

PASSIGN P2: 4663/NB
Similar to "4663" protocol, except that the plotter's block mode communications feature is not used.

OTHER COMMANDS FOR INITIALIZING PERIPHERAL PORTS

Besides the <Port-Assign> command, several other commands are needed to initialize a peripheral port for use with a particular peripheral device. These commands are: <set-port-baud-rate>, <set-port-EOF-string>, <set-port-EOL-string>, <set-port-flagging-mode>, <set-port-parity>, and <set-port-stop-bits>.

All these commands (including <port-assign>) are remembered by the terminal even when it is turned off. Thus these commands normally need be issued only once, when the operator attaches the peripheral device to the terminal's peripheral port.

<Set-Port-Baud-Rate> Command

The SETUP mode name for the <set-port-baud-rate> command is PBAUD. For instance, in SETUP mode, the operator can type:

PBAUD P0: 300

This sets the transmit and receive baud rates at peripheral port zero to 300 bits per second. The equivalent escape-sequence command for this is:

<set-port-baud-rate: "P0:", 300>

= (ESC)(P)(B)
  <int: 300>
= (ESC)(P)(B)(R)(<) .

<Set-Port-EOF-String> Command

The <set-port-EOF-string> command sets the end-of-file string used at a peripheral port. If the peripheral port does not have a plotter protocol <port-assign>ed to it, then the port's end-of-file string, as set by this command, is appended to the end of any data sent to that port. Likewise, if the peripheral port is the source for a data transfer, the port's end-of-file string serves to mark the end of the data transfer in a <copy>, <load>, <spool>, or <port-copy> operation.

The SETUP mode name for <set-port-EOF-string> is PEOF. Thus, the operator can type the following command in SETUP mode:

PEOF P1: "/**/"

This sets peripheral port one's end-of-file string to the following four-character sequence: (/)(*)(*)(/). The escape-sequence version for this example is:

<set-port-EOF-string: "P1:", (47,42,42,47)>

= (ESC)(P)(E)
  <string: "P1:">
  <int-array: (47,42,42,47)
= (ESC)(P)(E)(3)(P)(1)(:)(4)(B)(?)(B)(:)(B)(:)(B)(?) .

<Set-Port-EOL-String> Command

The <set-port-EOL-string> command sets a peripheral port's "end-of-line string." The port end-of-line string is used only if the "4643" protocol has been <port-assign>ed to that port. If that is the case, every (CR) character which otherwise would be sent to the port is replaced with the port's end-of-line string.

For example, suppose you want to print a file directory on a printer attached to peripheral port one. The <directory> command produces text in which each line ends with (CR) — not (CR)(LF). So, you type the following commands in SETUP mode:

```
PEOL P1: "CRLF"
PASSIGN P1: 4643
DIRECTORY F0: TO P1:
```

Here, the PEOL command is the SETUP mode version of <set-port-EOL-string>. It assigns the character sequence (CR)(LF) as the end-of-line string for peripheral port one. The PASSIGN command assigns the "4643" protocol to that port, so that any (CR)s in the data will be replaced with (CR)(LF) sequences. Finally, the DIRECTORY command sends a directory of the files on disk drive zero the printer attached to peripheral port one.

The escape-sequence version of this PEOL command is as follows:

```
<set-port-EOL-string: "P1:", (13,10)>
= (ESC)(P)(M)
    <string: "P1:">
    <int-array: (13,10)>
= (ESC)(P)(M)(3)(P)(1)(:)(2)(=)(:)
```

If the data to be transferred to a peripheral port has (CR)(LF) sequences at the end of each line, then you can set the port's end-of-line string to just (CR). Alternatively, the PPORT protocol can be <port-assign>ed to the peripheral port.

<Set-Port-Flagging-Mode> Command

The <set-port-flagging-mode> command chooses the flagging mode used at a peripheral port, just as the <set-flagging-mode> command selects the flagging mode used for communicating with the host computer.

In SETUP mode, the name of the command is PFLAG, and the keyword options for the first parameter are NONE, DTR/CTS, and CHAR. If CHAR (for character flagging) is chosen, the operator can enter single character third and fourth parameters to indicate the "go" and "stop" characters. The following examples illustrate this:

PFLAG P0: NONE	Disables flagging at peripheral port zero.
PFLAG P1: DTR/CTS	Enables DTR/CTS flagging at peripheral port one. A peripheral device at this port indicates that it is ready to receive data by asserting the DTR (Data Terminal Ready) signal at the RS-232 peripheral port connector. The terminal indicates when it is ready to receive data from a peripheral by asserting CTS (Clear To Send) at the port connector.
PFLAG P2: CHAR D_1 D_3	Enables "character flagging" at peripheral port two. A peripheral device at this port indicates when it is not ready for data by sending a (DC3) character to the terminal. When it is ready for more data, it sends a (DC1) to the terminal. Likewise, when the terminal accepts data from the peripheral device, it uses (DC1) as a "go" character and (DC3) as a "stop" character.
PFLAG P2: CHAR	Same as above; if the "go" and "stop" characters are omitted from the command, (DC1) and (DC3) are used as defaults.

The escape-sequence syntax for the <set-port-flagging> command is as follows:

<set-port-flagging> = (ESC)(P)(F)
<string: *port-identifier*>
<int: *flagging-mode*>
<int: *"GO"-character*>
<int: *"STOP"-character*> .

The port-identifier <string> is either "P0:", "P1:", or "P2:". The flagging-mode <int> is either zero (no flagging), one (for character flagging), or two (for DTR/CTS flagging). The GO and STOP characters are represented by their numeric equivalents (range 0 to 127), sent in the <int> parameters.

For example, to enable CHAR flagging at peripheral port two, with (DC1) as the GO character (ASCII decimal equivalent of 17), and with (DC3) as the STOP character (decimal equivalent 19), the host computer could send the following escape sequence:

<set-port-flagging: "P2:", 1, 17, 19>

= (ESC)(P)(F)
<string: "P2:">
<int: 1>
<int: 17>
<int: 19>

= (ESC)(P)(F)(3)(P)(2)(:)(1)(A)(1)(A)(3) .

<Set-Port-Parity> and <Set-Port-Stop-Bits> Commands

The <set-port-parity> and <set-port-stop-bits> commands are described in detail in the 4110 Series Command Reference Manual. (The SETUP mode names for these commands are PPARITY and PBITS, respectively.) The commands set a peripheral port's parity and number of stop bits to conform with the requirements of a peripheral device attached to that port.

The following examples show how these commands might be used in SETUP mode:

PPARITY P0: NONE PBITS 1 8	Sets port zero so that characters at that port have eight data bits, no parity bit, and one stop bit.
PPARITY P0: EVEN PBITS 2 7	Sets port zero so that characters at that port have seven data bits, one parity bit using even parity, and two stop bits.

For more details, see the descriptions of the <set-port-parity> and <set-port-stop-bits> commands in the 4110 Series Command Reference Manual.

USING A PRINTER

INITIALIZING THE PORT

When the operator attaches a printer to one of the terminal's RS-232 peripheral ports, he should also type several SETUP mode commands to configure the peripheral port for use with that printer.

4641 Printer. Before connecting a TEKTRONIX 4641 printer to one of the terminal's RS-232 peripheral ports, you should check to be sure that the 4641 is equipped with Option 30 (RS-232 Interface). The internal switches on the 4641 Buffered Serial Interface Card must also be set correctly; Table 12-5 lists one possible switch configuration.

Table 12-5

4641 INTERFACE CARD SWITCH SETTINGS

Switch	Condition
S1	OFF
S2	ON
S3	OFF
S4	ON
S5	OFF
S6	ON
S7	ON
S8	OFF

The switch settings in Table 12-5 set the 4641 printer to operate at 9600 baud, with DTR flagging when the printer's buffer is full.

With the 4641's switches set as in Table 12-5, you can type the following SETUP mode commands to initialize the terminal's peripheral port number one:

PBAUD P1: 9600 — Sets port one for 9600 baud. If the printer is set for a different baud rate, then use that baud rate in this command. (See "Technical Data," in Part One of the 4641/4641-1 Operator's Manual for details about setting the printer's baud rate.)

PPARITY P1: HIGH
PBITS P1: 1 7

or

PPARITY P1: NONE
PBITS P1: 1 8

The first two commands set port one to send ASCII characters consisting of seven data bits, a parity bit (always one), and one stop bit. The second two commands will also work; they set the port for eight data bits, no parity bit, and one stop bit.

Both of these settings assume that the 4641 is set for "no parity." (See "Jumper Functions," in Part Two, Section 3 of the 4641 Operator's Manual for information on setting the 4641 for "no parity.")

PEOL P1: "$^{C}{}_{R}{}^{L}{}_{F}$" — Sets port one's end-of-line string to (CR)(LF). This assumes that the 4641's jumper W8 is installed, thereby disabling the 4641's "CR implies CR-LF" feature. (See "Jumper Functions," in Part Two, Section 3 of the 4641 Operator's Manual.)

PASSIGN P1: 4643 — Assigns the "4643" protocol to peripheral port one. Any (CR) characters in text sent to this port will be replaced with the port's end-of-line string: (CR)(LF).

PFLAG P1: DTR/CTS — Sets port one for "DTR/CTS" flagging. This is necessary to prevent data over-run when sending text to a printer at port one.

4642 Printer. The TEKTRONIX 4642 Printer is not compatible with Tektronix 4110 Series terminals.

<COPY> TO A PRINTER

Once the printer port has been initialized, the operator or the host computer can issue <copy> commands to transfer data to the printer. For example, the operator can create a file holding a directory of disk files, and then <copy> that file to a printer at port one, as follows:

```
DIRECTORY F0: TO F0:FILE1
COPY F0:FILE1 TO P1:
```

<DIRECTORY> TO A PRINTER

The following SETUP mode command causes a disk directory to go directly to a printer at peripheral port number one:

```
DIRECTORY F0: TO P1:
```

<SPOOL> TO A PRINTER

If you have much data to send to a printer, you may wish to use the <spool> command rather than <copy>. With the <spool> command, the data transfer occurs "in the background." That is, the terminal can be used for other purposes while it is <spool>ing data to a printer.

For example, the operator might type these two commands in SETUP mode:

```
DIRECTORY F0: TO F0:DIRECTORY
SPOOL F0:DIRECTORY TO P1:
```

Here, the DIRECTORY command creates a disk file named DIRECTORY, and the SPOOL command copies that file to a printer attached to port number one. Unlike the COPY command, the SPOOL command lets the operator use the terminal for other purposes while the file is being printed out.

The escape-sequence version of the <spool> command has the following syntax:

```
<spool> = (ESC)(J)(S)
          <string: source-specifier>
          <string: empty-string or "TO">
          <string: destination-specifier> .
```

For example, the escape-sequence version of the SETUP mode command, "SPOOL F0:DIRECTORY TO P1:" would be:

```
<spool: "F0:DIRECTORY", empty-string, "P1:">
= (ESC)(J)(S)
  <string: "F0:DIRECTORY">
  <string: empty-string>
  <string: "P1:">
= (ESC)(J)(S)(<)(F)(0)(:)(D)(I)(R)(E)(C)
  (T)(O)(R)(Y)(0)(3)(P)(1)(:) .
```

To abort a <spool> operation before it is finished, use the <stop-spooling> command. (The SETUP mode name for this command is STOP.)

```
<stop-spooling> = (ESC)(J)(E) .
```

For more information on the <spool> and <stop-spooling> commands, see their descriptions in the 4110 Series Command Reference Manual.

USING A PLOTTER

CONNECTING THE PLOTTER TO THE TERMINAL

When using a TEKTRONIX 4662 or 4663 Interactive Digital Plotter, connect the plotter's "modem" connector to one of the terminal's three RS-232 peripheral ports.

NOTE

Do not connect the plotter between the terminal's host port and the host computer. Although the plotter is designed to run in this configuration, the terminal design assumes that the plotter is connected to one of the RS-232 peripheral ports.

INITIALIZING THE PORT

To use a TEKTRONIX 4662 or 4663 Interactive Digital Plotter with the terminal's RS-232 peripheral port, you must set the plotter's switches and the peripheral port settings in a compatible way. For instance, if the plotter is set to operate at 1200 baud, then the peripheral port to which it is attached must also be set for 1200 baud.

4662 Plotter. Table 12-6 shows one way the switches on the TEKTRONIX 4662 Digital Plotter can be set for communicating with the terminal through the terminal's Option 10 Three Port Peripheral Interface.

Table 12-6

4662 PLOTTER SETTINGS

Switch	Setting
A	3
B	3
C	2
D	3

These switch positions set the 4662 as follows:

Copy mode.
CR implies CR-LF.
GIN terminator = CR.
Number of stop bits = 1.
No Parity.
Plotter device address = A
Baud rate = 1200.

With the 4662's switches set as in Table 12-6, the following SETUP mode commands will then initialize peripheral port zero for communicating with the plotter. The terminal remembers these settings even when turned off; thus the commands need only be issued once, when attaching the plotter to the terminal.

PBAUD P0: 1200	Sets port zero to communicate with the 4662 at 1200 bits per second.
PASSIGN P0: 4662	Assigns the standard "4662" device protocol to peripheral port zero. (If the 4662 is equipped with multiple pens — Option 31 — then the second parameter in this command should be "4662/MP" rather than "4662.")
PPARITY P0: NONE PBITS P0: 1 8	Sets peripheral port zero to send ASCII characters which have eight data bits, no parity bit, and one stop bit.
PFLAG P0: NONE	Disables DTR/CTS and character flagging at the peripheral port. (Instead, the plotter's block mode is used to prevent data overrun.)

4663 Plotter. Table 12-7 shows one way to set the 4663 plotter's parameter entry switches for use with a 4110 Series terminal.

Table 12-7

4663 PLOTTER SETTINGS

Parameter	Setting
Initial aspect ratio	4X:3Y
Initial axis orientation	Y vertical, X horizontal
Interface select	1 (RS-232 interface)
Initial command response format	3
Serial device address	A
Receive baud rate	9600
Transmit baud rate	9600
Transmit baud rate limit	Full speed
Character format	8 data bits, 1 stop bit
Receive parity/transmit parity	Ignore/logic zero
Communications control mode	Full duplex
Interface functions	CR generates LF = YES, DEL IGNORE = NO
Attention character	ESC
Output terminator	CR

Suppose the plotter switches have been set as shown in Table 12-7. In that case, the following SETUP mode commands initialize peripheral port zero in the terminal for proper communication with the plotter:

PBAUD P0: 9600	Sets peripheral port zero to communicate with the plotter at 9600 bits per second.
PASSIGN P0: 4663	Assigns the "4663" device protocol to peripheral port zero.
PPARITY P0: NONE PBITS P0: 1 8	Sets peripheral port zero to send ASCII characters which have eight data bits, no parity bit, and one stop bit.
PFLAG P0: NONE	Disables DTR/CTS and character flagging. (Instead of flagging, the plotter's block mode protocol is used to prevent data overrun.)

<SPOOL> TO A PLOTTER

One way to transfer data from the host computer to a plotter is with the <copy> command. If the plotter is connected to port zero, this command would be:

<copy: "HO:", "TO", "P0:">

However, this will tie up the terminal, and the host-to-terminal data communications line, for the entire time that it takes the plotter to draw the picture being sent.

If the terminal is equipped with a disk drive, a better technique would be to <copy> the data to a file on the disk drive, and then <spool> the data from that disk file to the plotter:

<copy: "HO:", "TO", "F0:TEMPFILE">
<spool: "F0:TEMPFILE", "TO", "P0:">

Here, the <copy> operation takes a comparatively short time, since data can be transferred to a disk file at a far faster rate than the plotter can plot that data. The <spool> command then sets up a "background" data transfer. The operator (and the host computer) can use the terminal for other purposes while the data is being <spool>ed from the disk file to the plotter.

The terminal's operator can gain similar benefits by using the SETUP mode SPOOL command. For instance, rather than typing

SAVE SEG 5 TO P0:

the operator might type

SAVE SEG 5 TO F0:TEMPFILE
SPOOL F0:TEMPFILE TO P0:

Once the SPOOL command has been typed, the operator can then type other commands to the terminal (or to the host), while the <spool> operation proceeds in the background.

NOTE

<Set-Graphtext-Size> and <Set-Graphtext-Rotation> are not translated to plotter commands by the plotter device drivers. If you have a file with either of these commands in it, make the file into a segment (if it does not already contain segments) by using the <Begin-Segment> command, loading the file, and the <End-Segment> command. Then save the segment to the plotter.

<SAVE>ING SEGMENTS TO A PLOTTER

Details of Data Sent When <Save>ing a Segment

When a segment is <save>d, the data sent to the <save> command's destination is arranged so as to permit a plotter to draw that segment correctly.

That is, the data sent to a <save> command's destination consists of 4110 Series escape-sequence commands, in the following order:

1. A <set-pivot-point> command, establishing the pivot point for the segment whose definition follows.
2. Commands to set segment attributes for "segment minus two." "Segment minus two" means "the default for segments not yet defined." Thus, these commands establish segment attributes for the segment whose definition follows. These attributes include visibility, display priority, writing mode, image transform parameters (x- and y-scale factors, rotation angle, position), highlighting, detectability, and the segment classes to which the segment belongs.
3. Finally comes the segment definition itself. This includes a <begin-segment> command, the contents of the segment (moves, draws, alphatext, graphtext, primitive attributes, etc.), and an <end-segment> command.

The segment attributes are sent before the segment definition (as attributes of "segment minus two"). That way, the plotter "knows" these attributes before it draws the segment.

Consequently, if you <save> a segment to a disk file, and then later <load> that file, you also <load> attributes for "segment minus two." That is, <load>ing a segment also changes the attributes for "segment minus two."

Using the <Spool> Command

You can, of course, issue a <save> command to send a segment definition directly to a plotter. For instance, if a plotter is connected to port zero, the operator can type this SETUP mode command:

```
SAVE SEG 3 TO P0:
```

However, this ties up the terminal for the entire time it takes for the data to be sent to the plotter. A better way would be to use the <spool> command:

```
SAVE SEG 3 TO F0:SEGMENT3
SPOOL F0:SEGMENT3 TO P0:
```

<PLOT> COMMAND

The <plot> command is similar to the <save> command in that it saves segment definitions as 4110-series escape-sequence commands. There are, however, some important differences:

- The <plot> command saves all segments which are visible in the current view.
- The <plot> command also saves <set-window> and <set-viewport> commands for the current view.
- Data generated by the <plot> command causes a plotter to draw only the contents of the current window, in a part of the plotter surface which corresponds to the terminal's current viewport.

Effect of <Plot> Command's Window-Viewport Transform

The fact that the <plot> command performs a window-viewport transform has several effects.

Definitions. A "<plot> image" is a picture drawn on the plotter in response to data generated by a <plot> command. Likewise, a "<save> image" is a picture drawn on the plotter in response to data generated by a <save> command. The "plotter viewport" is the part of the plotter surface on which the plotter is set up to draw images. (You can use the plotter's "locate lower left" and "locate upper right" switches to find the lower left and upper right corners of the plotter viewport.)

<Save> Image. When an image is drawn on the plotter in response to the <save> command, the lower left corner of the plotter viewport corresponds to the point (0,0) in terminal space. The upper right corner of the plotter viewport corresponds to the point (4095,3071) in terminal space.

Graphic Input From a Plotter. Likewise, when graphic input is done using the plotter as the graphic input device, the lower left corner of the plotter viewport corresponds to the point (0,0) in terminal space, and the upper right corner of the plotter viewport corresponds to the point (4095,3071) in terminal space.

<Plot> Image. However, when an image in drawn on the plotter in response to a <plot> command, the corners of the plotter viewport correspond to the points (0,0) and (4095,3071) *in 4113 normalized screen coordinates*. This is *not* the same as the points (0,0) and (4095,3071) in *terminal space*.

That is, a <plot> image shows the effect of the terminal's window-viewport transform and the location of the current viewport on the terminal's screen. This is not the case with <save> images or with graphic input from the plotter.

Differences Between <Plot> Images and <Save> Images. For compatibility with earlier TEKTRONIX terminals, on power-up the 4113's default window extends from (0,0) to (4095,3127) in terminal space, even though its viewport extends from (0,0) to (4095,-3071) in normalized screen coordinates. Thus, the terminal's default view is slightly distorted, and a <plot> image based on that view will also be slightly distorted. With the default view, a <plot> image and a <save> image of the same segment pass through slightly different points on the plotter surface. In the <save> image, the plotter viewport corresponds to the region from (0,0) to (4095,3071) in terminal space. In the <plot> image for the default view, the plotter viewport corresponds to the region from (0,0) to (4095,3127) in terminal space.

Precautions For Graphic Input. If you plan to draw an image on the plotter, and then do graphic input from the plotter while referring to that image, then you should do one of the following:

- Draw the image on the plotter using the <save> command, not the plot command.
- Alternatively, you can set the current view's window and viewport to both extend from (0,0) to (4095,-3071). With the window and viewport set that way, you can use the <plot> command to draw the image.

Either of these precautions will ensure that the coordinate system used when drawing the image on the plotter surface is the same as the coordinate system used for digitizing graphic data from the plotter.

Issuing the <Plot> Command

In SETUP mode, the operator types the PLOT command as follows:

PLOT TO P0:

or

PLOT TO F0:THISVIEW
SPOOL F0:THISVIEW TO P0:

The first example <plot>s all visible segments directly to the plotter attached at peripheral port zero. The second example is more efficient of the operator's time: it saves all visible segments as escape-sequence commands in the file F0:THISVIEW, and then <spool>s the data in that file to the plotter at port zero. Once the SPOOL command has been typed, the operator can use the terminal for other purposes while the data is transferred to the plotter.

The escape-sequence syntax for the <plot> command is as follows:

<plot> = (ESC)(P)(L)
<string: empty-string or "TO">
<string: *destination-specifier*> .

<PORT-COPY> COMMAND

The commands described so far for drawing pictures on an external plotter all take advantage of the terminal's capability of translating 4110 Series escape-sequence commands to equivalent commands for the plotter.

However, you may wish to control the plotter directly from a host program, without the terminal's firmware intervening. The <port-copy> command provides this capability.

The <port-copy> command establishes a bidirectional data path between the host computer and an RS-232 peripheral port, or between two RS-232 peripheral ports. The command has this syntax:

<port-copy> = (ESC)(P)(C)
<string: *source-specifier*>
<string: empty-string or "TO">
<string: *destination-specifier*> .

Here, the "source" and "destination" are the two ends of the data path. For instance, the following command establishes a bidirectional data path between the host computer and peripheral port one:

<port-copy: "HO:", empty-string, "P1:">

= (ESC)(P)(C)
<string: "HO:">
<string: empty-string>
<string: "P1:">

= (ESC)(P)(C)(3)(H)(O)(:)(0)(3)(P)(1)(:) .

The data path remains in effect (that is, the terminal continues to execute the <port-copy> command) until either the "source" or the "destination" sends an end-of-file string. (If the terminal's block mode is being used at the host port, then the host terminates the <port-copy> by sending a block with the end-of-file bit set.) At the host port, the "end-of-file string" is the <EOF-string> set by the most recent <set-EOF-string> command. At an RS-232 peripheral port, the "end-of-file string" is the string set for that port by a <set-port-EOF-string> command.

For an RS-232 peripheral port to be valid as a source or destination in the <port-copy> command, a <port-assign> command must have assigned the "PPORT" device protocol to that port.

For more details on the <port-copy> command, see the description of that command in the 4110 Series Command Reference Manual.

USING OTHER RS-232 DEVICES

You can use the terminal's Option 10 Three Port Peripheral Interface with other RS-232 devices than those described in this section. Of course, you must be sure to set the port parameters in a way which is compatible with those devices.

In particular, the PPORT (general purpose peripheral port) device protocol should be <port-assign>ed to any port which is not being used with plotters or printers.

The other port settings (port baud rate, port parity, etc.) will depend on the requirements of the particular peripheral device.

USING A COLOR COPIER

Option 09, 4690 Color Copiers Interface, lets you make color hardcopies on a 4690 Series Color Graphics Copier from your terminal. This interface includes several commands for controlling a color copier and provides two additional device specifiers.

The commands included in Option 09 are:

<report-color-hardcopy-status>
<reserve-copier>
<select-hardcopy-interface>
<set-image-orientation>
<set-number-of-copies>

These commands are explained in detail in the 4110 Series Command Reference.

The two additional device specifiers provided by Option 09 are HC: and SC:. The HC: specifier indicates the color hard copier interface. You can specify this device only as a destination for data (this device cannot send data). The SC: specifier indicates a "pseudodevice" (an area in the terminal's memory) that contains the screen's contents. This pseudodevice can serve only as a source of data (you cannot send data to SC:).

You can use these device specifiers to store a screen display in a disk file and later send the stored display to the color hard copier. First, issue a <copy> command that specifies SC: as the source and a disk file as the destination. Later, to produce a color hardcopy, specify the disk file as the source and HC: as the destination in a <spool> command.

Sending data to HC: from a source other than a file that was copied from SC: will probably produce an error condition. Although you are allowed to issue a <copy> or <spool> command that specifies SC: as the source and HC: as the destination, there is little reason to so. It is more straightforward to produce a color hardcopy with the <hardcopy> command or by pressing the hardcopy key.

Section 13

USING 4010-SERIES GRAPHICS PROGRAMS

RUNNING EXISTING 4010 SERIES PROGRAMS

Programs written for earlier TEKTRONIX terminals (4010 Series) were written without a dialog area in existence. They assume alphatext will be output beginning at the current graphic cursor position. This is not the case on the newer 4113 terminal if the dialog area is enabled. To run earlier programs on the 4113, without modifying those programs, the 4113's dialog area must be disabled. (See the descriptions of the <enable-dialog-area> command in Section 4 of this manual and in the 4110 Series Command Reference Manual.)

NOTE

On the other hand, if you want to take advantage of, and allow for, the 4113's dialog area, you can very easily modify earlier programs to do so. This involves making sure the program uses the <graphic-text> command to output all alphatext destined for the graphics area. (See the descriptions of the <graphic-text> command in Section 7 of this manual and in the 4110 Series Command Reference Manual.)

EMULATING 4010 SERIES TERMINALS WITH 4953/4954 GRAPHICS TABLETS

The 4113 can emulate 4010 Series terminals with accessory graphics tablets. To do this, the 4113 must have a graphic tablet installed (Option 13 or 14). With Option 13 or 14 installed, the 4113 can emulate a TEKTRONIX 4010 Series terminal which has an accessory TEKTRONIX 4953 or 4954 Graphics Tablet.

There are four commands used for the emulation:

- <Enable-4953-Tablet-GIN>
- <Disable-4953-Tablet-GIN>
- <Set-Tablet-Header-Characters>
- <Set-Tablet-Status-Strap>

For descriptions of these commands, see the 4110 Series Command Reference Manual. The description of the <enable-4953-tablet-GIN> command is especially important; it includes a table showing the commands which you can give the 4113 in order to emulate strap settings on the 4953/4954 Tablet Controller board.

Appendix A

ASCII CHARTS

This appendix includes a standard ASCII code chart and additional ASCII code charts which define the specific characters used as parameters (indicated by unshaded areas).

The code charts are:

Table	Description
A-1	ASCII Code Chart
A-2	Characters Used in <Char> Parameters
A-3	Characters Used in <Int> and <Int+> Parameters
A-4	Characters Used in <Int-Report> Parameters
A-5	Characters Used in <Xy> Parameters
A-6	Characters Used in <Xy-Report> Parameters

Table A-1

ASCII (ISO-7-US) CODE CHART

B7 B6 B5 / BITS B4 B3 B2 B1	Ø Ø Ø	Ø Ø 1	Ø 1 Ø	Ø 1 1	1 Ø Ø	1 Ø 1	1 1 Ø	1 1 1
	CONTROL		FIGURES		UPPERCASE		LOWERCASE	
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 A-2

CHARACTERS USED IN <CHAR> PARAMETERS

B7 B6 B5 / BITS B4 B3 B2 B1	Ø Ø Ø	Ø Ø 1	Ø 1 Ø	Ø 1 1	1 Ø Ø	1 Ø 1	1 1 Ø	1 1 1
	CONTROL		FIGURES		UPPERCASE		LOWERCASE	
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 A-3

CHARACTERS USED IN <INT> AND <INT+> PARAMETERS

<HiI> Characters

B7 B6 B5 / BITS / B4 B3 B2 B1	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
	CONTROL		FIGURES		UPPERCASE		LOWERCASE	
0 0 0 0	NUL 0	DLE 16	SP 32	0 48	@ 64	P 80	\ 96	p 112
0 0 0 1	SOH 1	DC1 17	! 33	1 49	A 65	Q 81	a 97	q 113
0 0 1 0	STX 2	DC2 18	" 34	2 50	B 66	R 82	b 98	r 114
0 0 1 1	ETX 3	DC3 19	# 35	3 51	C 67	S 83	c 99	s 115
0 1 0 0	EOT 4	DC4 20	$ 36	4 52	D 68	T 84	d 100	t 116
0 1 0 1	ENQ 5	NAK 21	% 37	5 53	E 69	U 85	e 101	u 117
0 1 1 0	ACK 6	SYN 22	& 38	6 54	F 70	V 86	f 102	v 118
0 1 1 1	BEL 7	ETB 23	' 39	7 55	G 71	W 87	g 103	w 119
1 0 0 0	BS 8	CAN 24	(40	8 56	H 72	X 88	h 104	x 120
1 0 0 1	HT 9	EM 25	) 41	9 57	I 73	Y 89	i 105	y 121
1 0 1 0	LF 10	SUB 26	* 42	: 58	J 74	Z 90	j 106	z 122
1 0 1 1	VT 11	ESC 27	+ 43	; 59	K 75	[91	k 107	{ 123
1 1 0 0	FF 12	FS 28	, 44	< 60	L 76	\ 92	l 108	|* 124
1 1 0 1	CR 13	GS 29	- 45	= 61	M 77	] 93	m 109	} 125
1 1 1 0	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

<LoI> Characters

B7 B6 B5 / BITS / B4 B3 B2 B1	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
	CONTROL		FIGURES		UPPERCASE		LOWERCASE	
0 0 0 0	NUL 0	DLE 16	SP 32	0 48	@ 64	P 80	\ 96	p 112
0 0 0 1	SOH 1	DC1 17	! 33	1 49	A 65	Q 81	a 97	q 113
0 0 1 0	STX 2	DC2 18	" 34	2 50	B 66	R 82	b 98	r 114
0 0 1 1	ETX 3	DC3 19	# 35	3 51	C 67	S 83	c 99	s 115
0 1 0 0	EOT 4	DC4 20	$ 36	4 52	D 68	T 84	d 100	t 116
0 1 0 1	ENQ 5	NAK 21	% 37	5 53	E 69	U 85	e 101	u 117
0 1 1 0	ACK 6	SYN 22	& 38	6 54	F 70	V 86	f 102	v 118
0 1 1 1	BEL 7	ETB 23	' 39	7 55	G 71	W 87	g 103	w 119
1 0 0 0	BS 8	CAN 24	(40	8 56	H 72	X 88	h 104	x 120
1 0 0 1	HT 9	EM 25	) 41	9 57	I 73	Y 89	i 105	y 121
1 0 1 0	LF 10	SUB 26	* 42	: 58	J 74	Z 90	j 106	z 122
1 0 1 1	VT 11	ESC 27	+ 43	; 59	K 75	[91	k 107	{ 123
1 1 0 0	FF 12	FS 28	, 44	< 60	L 76	\ 92	l 108	|* 124
1 1 0 1	CR 13	GS 29	- 45	= 61	M 77	] 93	m 109	} 125
1 1 1 0	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

Table A-4

CHARACTERS USED IN <INT-REPORT> PARAMETERS

<Hil-Report> Characters

B7 B6 B5 / BITS B4 B3 B2 B1	Ø Ø Ø	Ø Ø 1	Ø 1 Ø	Ø 1 1	1 Ø Ø	1 Ø 1	1 1 Ø	1 1 1
	CONTROL		FIGURES		UPPERCASE		LOWERCASE	
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

<Lol-Report> Characters

B7 B6 B5 / BITS B4 B3 B2 B1	Ø Ø Ø	Ø Ø 1	Ø 1 Ø	Ø 1 1	1 Ø Ø	1 Ø 1	1 1 Ø	1 1 1
	CONTROL		FIGURES		UPPERCASE		LOWERCASE	
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

Table A-5

CHARACTERS USED IN <XY> PARAMETERS

<HiY>,<HiX> Characters

B7 B6 B5 / BITS B4 B3 B2 B1	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
	CONTROL		FIGURES		UPPERCASE		LOWERCASE	
0 0 0 0	NUL 0	DLE 16	SP 32	0 48	@ 64	P 80	\ 96	p 112
0 0 0 1	SOH 1	DC1 17	! 33	1 49	A 65	Q 81	a 97	q 113
0 0 1 0	STX 2	DC2 18	" 34	2 50	B 66	R 82	b 98	r 114
0 0 1 1	ETX 3	DC3 19	# 35	3 51	C 67	S 83	c 99	s 115
0 1 0 0	EOT 4	DC4 20	$ 36	4 52	D 68	T 84	d 100	t 116
0 1 0 1	ENQ 5	NAK 21	% 37	5 53	E 69	U 85	e 101	u 117
0 1 1 0	ACK 6	SYN 22	& 38	6 54	F 70	V 86	f 102	v 118
0 1 1 1	BEL 7	ETB 23	/ 39	7 55	G 71	W 87	g 103	w 119
1 0 0 0	BS 8	CAN 24	(40	8 56	H 72	X 88	h 104	x 120
1 0 0 1	HT 9	EM 25	) 41	9 57	I 73	Y 89	i 105	y 121
1 0 1 0	LF 10	SUB 26	* 42	: 58	J 74	Z 90	j 106	z 122
1 0 1 1	VT 11	ESC 27	+ 43	; 59	K 75	[91	k 107	{ 123
1 1 0 0	FF 12	FS 28	, 44	< 60	L 76	\ 92	l 108	|* 124
1 1 0 1	CR 13	GS 29	- 45	= 61	M 77	] 93	m 109	} 125
1 1 1 0	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

<LoY>,<Extra> Characters

B7 B6 B5 / BITS B4 B3 B2 B1	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
	CONTROL		FIGURES		UPPERCASE		LOWERCASE	
0 0 0 0	NUL 0	DLE 16	SP 32	0 48	@ 64	P 80	\ 96	p 112
0 0 0 1	SOH 1	DC1 17	! 33	1 49	A 65	Q 81	a 97	q 113
0 0 1 0	STX 2	DC2 18	" 34	2 50	B 66	R 82	b 98	r 114
0 0 1 1	ETX 3	DC3 19	# 35	3 51	C 67	S 83	c 99	s 115
0 1 0 0	EOT 4	DC4 20	$ 36	4 52	D 68	T 84	d 100	t 116
0 1 0 1	ENQ 5	NAK 21	% 37	5 53	E 69	U 85	e 101	u 117
0 1 1 0	ACK 6	SYN 22	& 38	6 54	F 70	V 86	f 102	v 118
0 1 1 1	BEL 7	ETB 23	/ 39	7 55	G 71	W 87	g 103	w 119
1 0 0 0	BS 8	CAN 24	(40	8 56	H 72	X 88	h 104	x 120
1 0 0 1	HT 9	EM 25	) 41	9 57	I 73	Y 89	i 105	y 121
1 0 1 0	LF 10	SUB 26	* 42	: 58	J 74	Z 90	j 106	z 122
1 0 1 1	VT 11	ESC 27	+ 43	; 59	K 75	[91	k 107	{ 123
1 1 0 0	FF 12	FS 28	, 44	< 60	L 76	\ 92	l 108	|* 124
1 1 0 1	CR 13	GS 29	- 45	= 61	M 77	] 93	m 109	} 125
1 1 1 0	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

<LoX> Characters

B7 B6 B5 / BITS B4 B3 B2 B1	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
	CONTROL		FIGURES		UPPERCASE		LOWERCASE	
0 0 0 0	NUL 0	DLE 16	SP 32	0 48	@ 64	P 80	\ 96	p 112
0 0 0 1	SOH 1	DC1 17	! 33	1 49	A 65	Q 81	a 97	q 113
0 0 1 0	STX 2	DC2 18	" 34	2 50	B 66	R 82	b 98	r 114
0 0 1 1	ETX 3	DC3 19	# 35	3 51	C 67	S 83	c 99	s 115
0 1 0 0	EOT 4	DC4 20	$ 36	4 52	D 68	T 84	d 100	t 116
0 1 0 1	ENQ 5	NAK 21	% 37	5 53	E 69	U 85	e 101	u 117
0 1 1 0	ACK 6	SYN 22	& 38	6 54	F 70	V 86	f 102	v 118
0 1 1 1	BEL 7	ETB 23	/ 39	7 55	G 71	W 87	g 103	w 119
1 0 0 0	BS 8	CAN 24	(40	8 56	H 72	X 88	h 104	x 120
1 0 0 1	HT 9	EM 25	) 41	9 57	I 73	Y 89	i 105	y 121
1 0 1 0	LF 10	SUB 26	* 42	: 58	J 74	Z 90	j 106	z 122
1 0 1 1	VT 11	ESC 27	+ 43	; 59	K 75	[91	k 107	{ 123
1 1 0 0	FF 12	FS 28	, 44	< 60	L 76	\ 92	l 108	|* 124
1 1 0 1	CR 13	GS 29	- 45	= 61	M 77	] 93	m 109	} 125
1 1 1 0	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

Table A-6

CHARACTERS USED IN <XY-REPORT> PARAMETERS

<HiY-Report>, <Extra-Report>, <LoY-Report>, <HiX-Report>, and <LoX-Report> Characters

B7 B6 B5 / BITS B4 B3 B2 B1	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
	CONTROL		FIGURES		UPPERCASE		LOWERCASE	
0 0 0 0	NUL 0	DLE 16	SP 32	0 48	@ 64	P 80	\ 96	p 112
0 0 0 1	SOH 1	DC1 17	! 33	1 49	A 65	Q 81	a 97	q 113
0 0 1 0	STX 2	DC2 18	" 34	2 50	B 66	R 82	b 98	r 114
0 0 1 1	ETX 3	DC3 19	# 35	3 51	C 67	S 83	c 99	s 115
0 1 0 0	EOT 4	DC4 20	$ 36	4 52	D 68	T 84	d 100	t 116
0 1 0 1	ENQ 5	NAK 21	% 37	5 53	E 69	U 85	e 101	u 117
0 1 1 0	ACK 6	SYN 22	& 38	6 54	F 70	V 86	f 102	v 118
0 1 1 1	BEL 7	ETB 23	' 39	7 55	G 71	W 87	g 103	w 119
1 0 0 0	BS 8	CAN 24	(40	8 56	H 72	X 88	h 104	x 120
1 0 0 1	HT 9	EM 25	) 41	9 57	I 73	Y 89	i 105	y 121
1 0 1 0	LF 10	SUB 26	* 42	: 58	J 74	Z 90	j 106	z 122
1 0 1 1	VT 11	ESC 27	+ 43	; 59	K 75	[91	k 107	{ 123
1 1 0 0	FF 12	FS 28	, 44	< 60	L 76	\ 92	l 108	| * 124
1 1 0 1	CR 13	GS 29	- 45	= 61	M 77	] 93	m 109	} 125
1 1 1 0	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

Appendix B

EXAMPLES OF <INT> PARAMETERS

Table B-1 lists <int> parameters for integers between −1049 and + 1049.

Table B-1

REPRESENTATION OF NUMBERS AS <INT> PARAMETERS

n	<int : n>	−n	<int : −n>	n	<int : n>	−n	<int : −n>	n	<int : n>	−n	<int : −n>
0	(0)	−0	(SP)	50	(C)(2)	−50	(C)(")	100	(F)(4)	−100	(F)($)
1	(1)	−1	(!)	51	(C)(3)	−51	(C)(#)	101	(F)(5)	−101	(F)(%)
2	(2)	−2	(")	52	(C)(4)	−52	(C)($)	102	(F)(6)	−102	(F)(&)
3	(3)	−3	(#)	53	(C)(5)	−53	(C)(%)	103	(F)(7)	−103	(F)(')
4	(4)	−4	($)	54	(C)(6)	−54	(C)(&)	104	(F)(8)	−104	(F)("(")
5	(5)	−5	(%)	55	(C)(7)	−55	(C)(')	105	(F)(9)	−105	(F)(")")
6	(6)	−6	(&)	56	(C)(8)	−56	(C)("(")	106	(F)(:)	−106	(F)(*)
7	(7)	−7	(')	57	(C)(9)	−57	(C)(")")	107	(F)(;)	−107	(F)(+)
8	(8)	−8	("(")	58	(C)(:)	−58	(C)(*)	108	(F)(<)	−108	(F)(,)
9	(9)	−9	(")")	59	(C)(;)	−59	(C)(+)	109	(F)(=)	−109	(F)(-)
10	(:)	−10	(*)	60	(C)(<)	−60	(C)(,)	110	(F)(>)	−110	(F)(.)
11	(;)	−11	(+)	61	(C)(=)	−61	(C)(-)	111	(F)(?)	−111	(F)(/)
12	(<)	−12	(,)	62	(C)(>)	−62	(C)(.)	112	(G)(0)	−112	(G)(SP)
13	(=)	−13	(-)	63	(C)(?)	−63	(C)(/)	113	(G)(1)	−113	(G)(!)
14	(>)	−14	(.)	64	(D)(0)	−64	(D)(SP)	114	(G)(2)	−114	(G)(")
15	(?)	−15	(/)	65	(D)(1)	−65	(D)(!)	115	(G)(3)	−115	(G)(#)
16	(A)(0)	−16	(A)(SP)	66	(D)(2)	−66	(D)(")	116	(G)(4)	−116	(G)($)
17	(A)(1)	−17	(A)(!)	67	(D)(3)	−67	(D)(#)	117	(G)(5)	−117	(G)(%)
18	(A)(2)	−18	(A)(")	68	(D)(4)	−68	(D)($)	118	(G)(6)	−118	(G)(&)
19	(A)(3)	−19	(A)(#)	69	(D)(5)	−69	(D)(%)	119	(G)(7)	−119	(G)(')
20	(A)(4)	−20	(A)($)	70	(D)(6)	−70	(D)(&)	120	(G)(8)	−120	(G)("(")
21	(A)(5)	−21	(A)(%)	71	(D)(7)	−71	(D)(')	121	(G)(9)	−121	(G)(")")
22	(A)(6)	−22	(A)(&)	72	(D)(8)	−72	(D)("(")	122	(G)(:)	−122	(G)(*)
23	(A)(7)	−23	(A)(')	73	(D)(9)	−73	(D)(")")	123	(G)(;)	−123	(G)(+)
24	(A)(8)	−24	(A)("(")	74	(D)(:)	−74	(D)(*)	124	(G)(<)	−124	(G)(,)
25	(A)(9)	−25	(A)(")")	75	(D)(;)	−75	(D)(+)	125	(G)(=)	−125	(G)(-)
26	(A)(:)	−26	(A)(*)	76	(D)(<)	−76	(D)(,)	126	(G)(>)	−126	(G)(.)
27	(A)(;)	−27	(A)(+)	77	(D)(=)	−77	(D)(-)	127	(G)(?)	−127	(G)(/)
28	(A)(<)	−28	(A)(,)	78	(D)(>)	−78	(D)(.)	128	(H)(0)	−128	(H)(SP)
29	(A)(=)	−29	(A)(-)	79	(D)(?)	−79	(D)(/)	129	(H)(1)	−129	(H)(!)
30	(A)(>)	−30	(A)(.)	80	(E)(0)	−80	(E)(SP)	130	(H)(2)	−130	(H)(")
31	(A)(?)	−31	(A)(/)	81	(E)(1)	−81	(E)(!)	131	(H)(3)	−131	(H)(#)
32	(B)(0)	−32	(B)(SP)	82	(E)(2)	−82	(E)(")	132	(H)(4)	−132	(H)($)
33	(B)(1)	−33	(B)(!)	83	(E)(3)	−83	(E)(#)	133	(H)(5)	−133	(H)(%)
34	(B)(2)	−34	(B)(")	84	(E)(4)	−84	(E)($)	134	(H)(6)	−134	(H)(&)
35	(B)(3)	−35	(B)(#)	85	(E)(5)	−85	(E)(%)	135	(H)(7)	−135	(H)(')
36	(B)(4)	−36	(B)($)	86	(E)(6)	−86	(E)(&)	136	(H)(8)	−136	(H)("(")
37	(B)(5)	−37	(B)(%)	87	(E)(7)	−87	(E)(')	137	(H)(9)	−137	(H)(")")
38	(B)(6)	−38	(B)(&)	88	(E)(8)	−88	(E)("(")	138	(H)(:)	−138	(H)(*)
39	(B)(7)	−39	(B)(')	89	(E)(9)	−89	(E)(")")	139	(H)(;)	−139	(H)(+)
40	(B)(8)	−40	(B)("(")	90	(E)(:)	−90	(E)(*)	140	(H)(<)	−140	(H)(,)
41	(B)(9)	−41	(B)(")")	91	(E)(;)	−91	(E)(+)	141	(H)(=)	−141	(H)(-)
42	(B)(:)	−42	(B)(*)	92	(E)(<)	−92	(E)(,)	142	(H)(>)	−142	(H)(.)
43	(B)(;)	−43	(B)(+)	93	(E)(=)	−93	(E)(-)	143	(H)(?)	−143	(H)(/)
44	(B)(<)	−44	(B)(,)	94	(E)(>)	−94	(E)(.)	144	(I)(0)	−144	(I)(SP)
45	(B)(=)	−45	(B)(-)	95	(E)(?)	−95	(E)(/)	145	(I)(1)	−145	(I)(!)
46	(B)(>)	−46	(B)(.)	96	(F)(0)	−96	(F)(SP)	146	(I)(2)	−146	(I)(")
47	(B)(?)	−47	(B)(/)	97	(F)(1)	−97	(F)(!)	147	(I)(3)	−147	(I)(#)
48	(C)(0)	−48	(C)(SP)	98	(F)(2)	−98	(F)(")	148	(I)(4)	−148	(I)($)
49	(C)(1)	−49	(C)(!)	99	(F)(3)	−99	(F)(#)	149	(I)(5)	−149	(I)(%)

Table B-1, Continued

REPRESENTATION OF NUMBERS AS <INT> PARAMETERS

n	<int : n>	−n	<int : −n>
150	(I)(6)	−150	(I)(&)
151	(I)(7)	−151	(I)(')
152	(I)(8)	−152	(I)(“(”)
153	(I)(9)	−153	(I)(“)”)
154	(I)(:)	−154	(I)(*)
155	(I)(;)	−155	(I)(+)
156	(I)(<)	−156	(I)(,)
157	(I)(=)	−157	(I)(-)
158	(I)(>)	−158	(I)(.)
159	(I)(?)	−159	(I)(/)
160	(J)(0)	−160	(J)(SP)
161	(J)(1)	−161	(J)(!)
162	(J)(2)	−162	(J)(”)
163	(J)(3)	−163	(J)(#)
164	(J)(4)	−164	(J)($)
165	(J)(5)	−165	(J)(%)
166	(J)(6)	−166	(J)(&)
167	(J)(7)	−167	(J)(')
168	(J)(8)	−168	(J)(“(”)
169	(J)(9)	−169	(J)(“)”)
170	(J)(:)	−170	(J)(*)
171	(J)(;)	−171	(J)(+)
172	(J)(<)	−172	(J)(,)
173	(J)(=)	−173	(J)(-)
174	(J)(>)	−174	(J)(.)
175	(J)(?)	−175	(J)(/)
176	(K)(0)	−176	(K)(SP)
177	(K)(1)	−177	(K)(!)
178	(K)(2)	−178	(K)(”)
179	(K)(3)	−179	(K)(#)
180	(K)(4)	−180	(K)($)
181	(K)(5)	−181	(K)(%)
182	(K)(6)	−182	(K)(&)
183	(K)(7)	−183	(K)(')
184	(K)(8)	−184	(K)(“(”)
185	(K)(9)	−185	(K)(“)”)
186	(K)(:)	−186	(K)(*)
187	(K)(;)	−187	(K)(+)
188	(K)(<)	−188	(K)(,)
189	(K)(=)	−189	(K)(-)
190	(K)(>)	−190	(K)(.)
191	(K)(?)	−191	(K)(/)
192	(L)(0)	−192	(L)(SP)
193	(L)(1)	−193	(L)(!)
194	(L)(2)	−194	(L)(”)
195	(L)(3)	−195	(L)(#)
196	(L)(4)	−196	(L)($)
197	(L)(5)	−197	(L)(%)
198	(L)(6)	−198	(L)(&)
199	(L)(7)	−199	(L)(')
200	(L)(8)	−200	(L)(“(”)
201	(L)(9)	−201	(L)(“)”)
202	(L)(:)	−202	(L)(*)
203	(L)(;)	−203	(L)(+)
204	(L)(<)	−204	(L)(,)
205	(L)(=)	−205	(L)(-)
206	(L)(>)	−206	(L)(.)
207	(L)(?)	−207	(L)(/)
208	(M)(0)	−208	(M)(SP)
209	(M)(1)	−209	(M)(!)
210	(M)(2)	−210	(M)(”)
211	(M)(3)	−211	(M)(#)
212	(M)(4)	−212	(M)($)
213	(M)(5)	−213	(M)(%)
214	(M)(6)	−214	(M)(&)
215	(M)(7)	−215	(M)(')
216	(M)(8)	−216	(M)(“(”)
217	(M)(9)	−217	(M)(“)”)
218	(M)(:)	−218	(M)(*)
219	(M)(;)	−219	(M)(+)
220	(M)(<)	−220	(M)(,)
221	(M)(=)	−221	(M)(-)
222	(M)(>)	−222	(M)(.)
223	(M)(?)	−223	(M)(/)
224	(N)(0)	−224	(N)(SP)
225	(N)(1)	−225	(N)(!)
226	(N)(2)	−226	(N)(”)
227	(N)(3)	−227	(N)(#)
228	(N)(4)	−228	(N)($)
229	(N)(5)	−229	(N)(%)
230	(N)(6)	−230	(N)(&)
231	(N)(7)	−231	(N)(')
232	(N)(8)	−232	(N)(“(”)
233	(N)(9)	−233	(N)(“)”)
234	(N)(:)	−234	(N)(*)
235	(N)(;)	−235	(N)(+)
236	(N)(<)	−236	(N)(,)
237	(N)(=)	−237	(N)(-)
238	(N)(>)	−238	(N)(.)
239	(N)(?)	−239	(N)(/)
240	(O)(0)	−240	(O)(SP)
241	(O)(1)	−241	(O)(!)
242	(O)(2)	−242	(O)(”)
243	(O)(3)	−243	(O)(#)
244	(O)(4)	−244	(O)($)
245	(O)(5)	−245	(O)(%)
246	(O)(6)	−246	(O)(&)
247	(O)(7)	−247	(O)(')
248	(O)(8)	−248	(O)(“(”)
249	(O)(9)	−249	(O)(“)”)
250	(O)(:)	−250	(O)(*)
251	(O)(;)	−251	(O)(+)
252	(O)(<)	−252	(O)(,)
253	(O)(=)	−253	(O)(-)
254	(O)(>)	−254	(O)(.)
255	(O)(?)	−255	(O)(/)
256	(P)(0)	−256	(P)(SP)
257	(P)(1)	−257	(P)(!)
258	(P)(2)	−258	(P)(”)
259	(P)(3)	−259	(P)(#)
260	(P)(4)	−260	(P)($)
261	(P)(5)	−261	(P)(%)
262	(P)(6)	−262	(P)(&)
263	(P)(7)	−263	(P)(')
264	(P)(8)	−264	(P)(“(”)
265	(P)(9)	−265	(P)(“)”)
266	(P)(:)	−266	(P)(*)
267	(P)(;)	−267	(P)(+)
268	(P)(<)	−268	(P)(,)
269	(P)(=)	−269	(P)(-)
270	(P)(>)	−270	(P)(.)
271	(P)(?)	−271	(P)(/)
272	(Q)(0)	−272	(Q)(SP)
273	(Q)(1)	−273	(Q)(!)
274	(Q)(2)	−274	(Q)(”)
275	(Q)(3)	−275	(Q)(#)
276	(Q)(4)	−276	(Q)($)
277	(Q)(5)	−277	(Q)(%)
278	(Q)(6)	−278	(Q)(&)
279	(Q)(7)	−279	(Q)(')
280	(Q)(8)	−280	(Q)(“(”)
281	(Q)(9)	−281	(Q)(“)”)
282	(Q)(:)	−282	(Q)(*)
283	(Q)(;)	−283	(Q)(+)
284	(Q)(<)	−284	(Q)(,)
285	(Q)(=)	−285	(Q)(-)
286	(Q)(>)	−286	(Q)(.)
287	(Q)(?)	−287	(Q)(/)
288	(R)(0)	−288	(R)(SP)
289	(R)(1)	−289	(R)(!)
290	(R)(2)	−290	(R)(”)
291	(R)(3)	−291	(R)(#)
292	(R)(4)	−292	(R)($)
293	(R)(5)	−293	(R)(%)
294	(R)(6)	−294	(R)(&)
295	(R)(7)	−295	(R)(')
296	(R)(8)	−296	(R)(“(”)
297	(R)(9)	−297	(R)(“)”)
298	(R)(:)	−298	(R)(*)
299	(R)(;)	−299	(R)(+)

Table B-1, Continued

REPRESENTATION OF NUMBERS AS <INT> PARAMETERS

n	<int : n>	−n	<int : −n>	n	<int : n>	−n	<int : −n>	n	<int : n>	−n	<int : −n>
300	(R)(<)	−300	(R)(,)	350	(U)(>)	−350	(U)(.)	400	(Y)(0)	−400	(Y)(SP)
301	(R)(=)	−301	(R)(-)	351	(U)(?)	−351	(U)(/)	401	(Y)(1)	−401	(Y)(!)
302	(R)(>)	−302	(R)(.)	352	(V)(0)	−352	(V)(SP)	402	(Y)(2)	−402	(Y)(")
303	(R)(?)	−303	(R)(/)	353	(V)(1)	−353	(V)(!)	403	(Y)(3)	−403	(Y)(#)
304	(S)(0)	−304	(S)(SP)	354	(V)(2)	−354	(V)(")	404	(Y)(4)	−404	(Y)($)
305	(S)(1)	−305	(S)(!)	355	(V)(3)	−355	(V)(#)	405	(Y)(5)	−405	(Y)(%)
306	(S)(2)	−306	(S)(")	356	(V)(4)	−356	(V)($)	406	(Y)(6)	−406	(Y)(&)
307	(S)(3)	−307	(S)(#)	357	(V)(5)	−357	(V)(%)	407	(Y)(7)	−407	(Y)(')
308	(S)(4)	−308	(S)($)	358	(V)(6)	−358	(V)(&)	408	(Y)(8)	−408	(Y)("(")
309	(S)(5)	−309	(S)(%)	359	(V)(7)	−359	(V)(')	409	(Y)(9)	−409	(Y)(")")
310	(S)(6)	−310	(S)(&)	360	(V)(8)	−360	(V)("(")	410	(Y)(:)	−410	(Y)(*)
311	(S)(7)	−311	(S)(')	361	(V)(9)	−361	(V)(")")	411	(Y)(;)	−411	(Y)(+)
312	(S)(8)	−312	(S)("(")	362	(V)(:)	−362	(V)(*)	412	(Y)(<)	−412	(Y)(,)
313	(S)(9)	−313	(S)(")")	363	(V)(;)	−363	(V)(+)	413	(Y)(=)	−413	(Y)(-)
314	(S)(:)	−314	(S)(*)	364	(V)(<)	−364	(V)(,)	414	(Y)(>)	−414	(Y)(.)
315	(S)(;)	−315	(S)(+)	365	(V)(=)	−365	(V)(-)	415	(Y)(?)	−415	(Y)(/)
316	(S)(<)	−316	(S)(,)	366	(V)(>)	−366	(V)(.)	416	(Z)(0)	−416	(Z)(SP)
317	(S)(=)	−317	(S)(-)	367	(V)(?)	−367	(V)(/)	417	(Z)(1)	−417	(Z)(!)
318	(S)(>)	−318	(S)(.)	368	(W)(0)	−368	(W)(SP)	418	(Z)(2)	−418	(Z)(")
319	(S)(?)	−319	(S)(/)	369	(W)(1)	−369	(W)(!)	419	(Z)(3)	−419	(Z)(#)
320	(T)(0)	−320	(T)(SP)	370	(W)(2)	−370	(W)(")	420	(Z)(4)	−420	(Z)($)
321	(T)(1)	−321	(T)(!)	371	(W)(3)	−371	(W)(#)	421	(Z)(5)	−421	(Z)(%)
322	(T)(2)	−322	(T)(")	372	(W)(4)	−372	(W)($)	422	(Z)(6)	−422	(Z)(&)
323	(T)(3)	−323	(T)(#)	373	(W)(5)	−373	(W)(%)	423	(Z)(7)	−423	(Z)(')
324	(T)(4)	−324	(T)($)	374	(W)(6)	−374	(W)(&)	424	(Z)(8)	−424	(Z)("(")
325	(T)(5)	−325	(T)(%)	375	(W)(7)	−375	(W)(')	425	(Z)(9)	−425	(Z)(")")
326	(T)(6)	−326	(T)(&)	376	(W)(8)	−376	(W)("(")	426	(Z)(:)	−426	(Z)(*)
327	(T)(7)	−327	(T)(')	377	(W)(9)	−377	(W)(")")	427	(Z)(;)	−427	(Z)(+)
328	(T)(8)	−328	(T)("(")	378	(W)(:)	−378	(W)(*)	428	(Z)(<)	−428	(Z)(,)
329	(T)(9)	−329	(T)(")")	379	(W)(;)	−379	(W)(+)	429	(Z)(=)	−429	(Z)(-)
330	(T)(:)	−330	(T)(*)	380	(W)(<)	−380	(W)(,)	430	(Z)(>)	−430	(Z)(.)
331	(T)(;)	−331	(T)(+)	381	(W)(=)	−381	(W)(-)	431	(Z)(?)	−431	(Z)(/)
332	(T)(<)	−332	(T)(,)	382	(W)(>)	−382	(W)(.)	432	([])(0)	−432	([])(SP)
333	(T)(=)	−333	(T)(-)	383	(W)(?)	−383	(W)(/)	433	([])(1)	−433	([])(!)
334	(T)(>)	−334	(T)(.)	384	(X)(0)	−384	(X)(SP)	434	([])(2)	−434	([])(")
335	(T)(?)	−335	(T)(/)	385	(X)(1)	−385	(X)(!)	435	([])(3)	−435	([])(#)
336	(U)(0)	−336	(U)(SP)	386	(X)(2)	−386	(X)(")	436	([])(4)	−436	([])($)
337	(U)(1)	−337	(U)(!)	387	(X)(3)	−387	(X)(#)	437	([])(5)	−437	([])(%)
338	(U)(2)	−338	(U)(")	388	(X)(4)	−388	(X)($)	438	([])(6)	−438	([])(&)
339	(U)(3)	−339	(U)(#)	389	(X)(5)	−389	(X)(%)	439	([])(7)	−439	([])(')
340	(U)(4)	−340	(U)($)	390	(X)(6)	−390	(X)(&)	440	([])(8)	−440	([])("(")
341	(U)(5)	−341	(U)(%)	391	(X)(7)	−391	(X)(')	441	([])(9)	−441	([])(")")
342	(U)(6)	−342	(U)(&)	392	(X)(8)	−392	(X)("(")	442	([])(:)	−442	([])(*)
343	(U)(7)	−343	(U)(')	393	(X)(9)	−393	(X)(")")	443	([])(;)	−443	([])(+)
344	(U)(8)	−344	(U)("(")	394	(X)(:)	−394	(X)(*)	444	([])(<)	−444	([])(,)
345	(U)(9)	−345	(U)(")")	395	(X)(;)	−395	(X)(+)	445	([])(=)	−445	([])(-)
346	(U)(:)	−346	(U)(*)	396	(X)(<)	−396	(X)(,)	446	([])(>)	−446	([])(.)
347	(U)(;)	−347	(U)(+)	397	(X)(=)	−397	(X)(-)	447	([])(?)	−447	([])(/)
348	(U)(<)	−348	(U)(,)	398	(X)(>)	−398	(X)(.)	448	(\)(0)	−448	(\)(SP)
349	(U)(=)	−349	(U)(-)	399	(X)(?)	−399	(X)(/)	449	(\)(1)	−449	(\)(!)

Table B-1, Continued

REPRESENTATION OF NUMBERS AS <INT> PARAMETERS

n	<int : n>	−n	<int : −n>
450	(\)(2)	−450	(\)(")
451	(\)(3)	−451	(\)(#)
452	(\)(4)	−452	(\)($)
453	(\)(5)	−453	(\)(%)
454	(\)(6)	−454	(\)(&)
455	(\)(7)	−455	(\)(')
456	(\)(8)	−456	(\)("(")
457	(\)(9)	−457	(\)(")")
458	(\)(:)	−458	(\)(*)
459	(\)(;)	−459	(\)(+)
460	(\)(<)	−460	(\)(,)
461	(\)(=)	−461	(\)(-)
462	(\)(>)	−462	(\)(.)
463	(\)(?)	−463	(\)(/)
464	([])(0)	−464	([])(SP)
465	([])(1)	−465	([])(!)
466	([])(2)	−466	([])(")
467	([])(3)	−467	([])(#)
468	([])(4)	−468	([])($)
469	([])(5)	−469	([])(%)
470	([])(6)	−470	([])(&)
471	([])(7)	−471	([])(')
472	([])(8)	−472	([])("(")
473	([])(9)	−473	([])(")")
474	([])(:)	−474	([])(*)
475	([])(;)	−475	([])(+)
476	([])(<)	−476	([])(,)
477	([])(=)	−477	([])(-)
478	([])(>)	−478	([])(.)
479	([])(?)	−479	([])(/)
480	(^)(0)	−480	(^)(SP)
481	(^)(1)	−481	(^)(!)
482	(^)(2)	−482	(^)(")
483	(^)(3)	−483	(^)(#)
484	(^)(4)	−484	(^)($)
485	(^)(5)	−485	(^)(%)
486	(^)(6)	−486	(^)(&)
487	(^)(7)	−487	(^)(')
488	(^)(8)	−488	(^)("(")
489	(^)(9)	−489	(^)(")")
490	(^)(:)	−490	(^)(*)
491	(^)(;)	−491	(^)(+)
492	(^)(<)	−492	(^)(,)
493	(^)(=)	−493	(^)(-)
494	(^)(>)	−494	(^)(.)
495	(^)(?)	−495	(^)(/)
496	(_)(0)	−496	(_)(SP)
497	(_)(1)	−497	(_)(!)
498	(_)(2)	−498	(_)(")
499	(_)(3)	−499	(_)(#)
500	(_)(4)	−500	(_)($)
501	(_)(5)	−501	(_)(%)
502	(_)(6)	−502	(_)(&)
503	(_)(7)	−503	(_)(')
504	(_)("(")	−504	(_)("(")
505	(_)(9)	−505	(_)(")")
506	(_)(:)	−506	(_)(*)
507	(_)(;)	−507	(_)(+)
508	(_)(<)	−508	(_)(,)
509	(_)(=)	−509	(_)(-)
510	(_)(>)	−510	(_)(.)
511	(_)(?)	−511	(_)(/)
512	(')(0)	−512	(')(SP)
513	(')(1)	−513	(')(!)
514	(')(2)	−514	(')(")
515	(')(3)	−515	(')(#)
516	(')(4)	−516	(')($)
517	(')(5)	−517	(')(%)
518	(')(6)	−518	(')(&)
519	(')(7)	−519	(')(')
520	(')(8)	−520	(')("(")
521	(')(9)	−521	(')(")")
522	(')(:)	−522	(')(*)
523	(')(;)	−523	(')(+)
524	(')(<)	−524	(')(,)
525	(')(=)	−525	(')(-)
526	(')(>)	−526	(')(.)
527	(')(?)	−527	(')(/)
528	(a)(0)	−528	(a)(SP)
529	(a)(1)	−529	(a)(!)
530	(a)(2)	−530	(a)(")
531	(a)(3)	−531	(a)(#)
532	(a)(4)	−532	(a)($)
533	(a)(5)	−533	(a)(%)
534	(a)(6)	−534	(a)(&)
535	(a)(7)	−535	(a)(')
536	(a)(8)	−536	(a)("(")
537	(a)(9)	−537	(a)(")")
538	(a)(:)	−538	(a)(*)
539	(a)(;)	−539	(a)(+)
540	(a)(<)	−540	(a)(,)
541	(a)(=)	−541	(a)(-)
542	(a)(>)	−542	(a)(.)
543	(a)(?)	−543	(a)(/)
544	(b)(0)	−544	(b)(SP)
545	(b)(1)	−545	(b)(!)
546	(b)(2)	−546	(b)(")
547	(b)(3)	−547	(b)(#)
548	(b)(4)	−548	(b)($)
549	(b)(5)	−549	(b)(%)
550	(b)(6)	−550	(b)(&)
551	(b)(7)	−551	(b)(')
552	(b)(8)	−552	(b)("(")
553	(b)(9)	−553	(b)(")")
554	(b)(:)	−554	(b)(*)
555	(b)(;)	−555	(b)(+)
556	(b)(<)	−556	(b)(,)
557	(b)(=)	−557	(b)(-)
558	(b)(>)	−558	(b)(.)
559	(b)(?)	−559	(b)(/)
560	(c)(0)	−560	(c)(SP)
561	(c)(1)	−561	(c)(!)
562	(c)(2)	−562	(c)(")
563	(c)(3)	−563	(c)(#)
564	(c)(4)	−564	(c)($)
565	(c)(5)	−565	(c)(%)
566	(c)(6)	−566	(c)(&)
567	(c)(7)	−567	(c)(')
568	(c)(8)	−568	(c)("(")
569	(c)(9)	−569	(c)(")")
570	(c)(:)	−570	(c)(*)
571	(c)(;)	−571	(c)(+)
572	(c)(<)	−572	(c)(,)
573	(c)(=)	−573	(c)(-)
574	(c)(>)	−574	(c)(.)
575	(c)(?)	−575	(c)(/)
576	(d)(0)	−576	(d)(SP)
577	(d)(1)	−577	(d)(!)
578	(d)(2)	−578	(d)(")
579	(d)(3)	−579	(d)(#)
580	(d)(4)	−580	(d)($)
581	(d)(5)	−581	(d)(%)
582	(d)(6)	−582	(d)(&)
583	(d)(7)	−583	(d)(')
584	(d)(8)	−584	(d)("(")
585	(d)(9)	−585	(d)(")")
586	(d)(:)	−586	(d)(*)
587	(d)(;)	−587	(d)(+)
588	(d)(<)	−588	(d)(,)
589	(d)(=)	−589	(d)(-)
590	(d)(>)	−590	(d)(.)
591	(d)(?)	−591	(d)(/)
592	(e)(0)	−592	(e)(SP)
593	(e)(1)	−593	(e)(!)
594	(e)(2)	−594	(e)(")
595	(e)(3)	−595	(e)(#)
596	(e)(4)	−596	(e)($)
597	(e)(5)	−597	(e)(%)
598	(e)(6)	−598	(e)(&)
599	(e)(7)	−599	(e)(')

Table B-1, Continued

REPRESENTATION OF NUMBERS AS <INT> PARAMETERS

n	<int : n>	−n	<int : −n>
600	(e)(8)	−600	(e)(")
601	(e)(9)	−601	(e)(")")
602	(e)(:)	−602	(e)(*)
603	(e)(;)	−603	(e)(+)
604	(e)(<)	−604	(e)(,)
605	(e)(=)	−605	(e)(-)
606	(e)(>)	−606	(e)(.)
607	(e)(?)	−607	(e)(/)
608	(f)(0)	−608	(f)(SP)
609	(f)(1)	−609	(f)(!)
610	(f)(2)	−610	(f)(")
611	(f)(3)	−611	(f)(#)
612	(f)(4)	−612	(f)($)
613	(f)(5)	−613	(f)(%)
614	(f)(6)	−614	(f)(&)
615	(f)(7)	−615	(f)(')
616	(f)(8)	−616	(f)("(")
617	(f)(9)	−617	(f)(")")
618	(f)(:)	−618	(f)(*)
619	(f)(;)	−619	(f)(+)
620	(f)(<)	−620	(f)(,)
621	(f)(=)	−621	(f)(-)
622	(f)(>)	−622	(f)(.)
623	(f)(?)	−623	(f)(/)
624	(g)(0)	−624	(g)(SP)
625	(g)(1)	−625	(g)(!)
626	(g)(2)	−626	(g)(")
627	(g)(3)	−627	(g)(#)
628	(g)(4)	−628	(g)($)
629	(g)(5)	−629	(g)(%)
630	(g)(6)	−630	(g)(&)
631	(g)(7)	−631	(g)(')
632	(g)(8)	−632	(g)("(")
633	(g)(9)	−633	(g)(")")
634	(g)(:)	−634	(g)(*)
635	(g)(;)	−635	(g)(+)
636	(g)(<)	−636	(g)(,)
637	(g)(=)	−637	(g)(-)
638	(g)(>)	−638	(g)(.)
639	(g)(?)	−639	(g)(/)
640	(h)(0)	−640	(h)(SP)
641	(h)(1)	−641	(h)(!)
642	(h)(2)	−642	(h)(")
643	(h)(3)	−643	(h)(#)
644	(h)(4)	−644	(h)($)
645	(h)(5)	−645	(h)(%)
646	(h)(6)	−646	(h)(&)
647	(h)(7)	−647	(h)(')
648	(h)(8)	−648	(h)("(")
649	(h)(9)	−649	(h)(")")
650	(h)(:)	−650	(h)(*)
651	(h)(;)	−651	(h)(+)
652	(h)(<)	−652	(h)(,)
653	(h)(=)	−653	(h)(-)
654	(h)(>)	−654	(h)(.)
655	(h)(?)	−655	(h)(/)
656	(i)(0)	−656	(i)(SP)
657	(i)(1)	−657	(i)(!)
658	(i)(2)	−658	(i)(")
659	(i)(3)	−659	(i)(#)
660	(i)(4)	−660	(i)($)
661	(i)(5)	−661	(i)(%)
662	(i)(6)	−662	(i)(&)
663	(i)(7)	−663	(i)(')
664	(i)(8)	−664	(i)("(")
665	(i)(9)	−665	(i)(")")
666	(i)(:)	−666	(i)(*)
667	(i)(;)	−667	(i)(+)
668	(i)(<)	−668	(i)(,)
669	(i)(=)	−669	(i)(-)
670	(i)(>)	−670	(i)(.)
671	(i)(?)	−671	(i)(/)
672	(j)(0)	−672	(j)(SP)
673	(j)(1)	−673	(j)(!)
674	(j)(2)	−674	(j)(")
675	(j)(3)	−675	(j)(#)
676	(j)(4)	−676	(j)($)
677	(j)(5)	−677	(j)(%)
678	(j)(6)	−678	(j)(&)
679	(j)(7)	−679	(j)(')
680	(j)(8)	−680	(j)("(")
681	(j)(9)	−681	(j)(")")
682	(j)(:)	−682	(j)(*)
683	(j)(;)	−683	(j)(+)
684	(j)(<)	−684	(j)(,)
685	(j)(=)	−685	(j)(-)
686	(j)(>)	−686	(j)(.)
687	(j)(?)	−687	(j)(/)
688	(k)(0)	−688	(k)(SP)
689	(k)(1)	−689	(k)(!)
690	(k)(2)	−690	(k)(")
691	(k)(3)	−691	(k)(#)
692	(k)(4)	−692	(k)($)
693	(k)(5)	−693	(k)(%)
694	(k)(6)	−694	(k)(&)
695	(k)(7)	−695	(k)(')
696	(k)(8)	−696	(k)("(")
697	(k)(9)	−697	(k)(")")
698	(k)(:)	−698	(k)(*)
699	(k)(;)	−699	(k)(+)
700	(k)(<)	−700	(k)(,)
701	(k)(=)	−701	(k)(-)
702	(k)(>)	−702	(k)(.)
703	(k)(?)	−703	(k)(/)
704	(l)(0)	−704	(l)(SP)
705	(l)(1)	−705	(l)(!)
706	(l)(2)	−706	(l)(")
707	(l)(3)	−707	(l)(#)
708	(l)(4)	−708	(l)($)
709	(l)(5)	−709	(l)(%)
710	(l)(6)	−710	(l)(&)
711	(l)(7)	−711	(l)(')
712	(l)(8)	−712	(l)("(")
713	(l)(9)	−713	(l)(")")
714	(l)(:)	−714	(l)(*)
715	(l)(;)	−715	(l)(+)
716	(l)(<)	−716	(l)(,)
717	(l)(=)	−717	(l)(-)
718	(l)(>)	−718	(l)(.)
719	(l)(?)	−719	(l)(/)
720	(m)(0)	−720	(m)(SP)
721	(m)(1)	−721	(m)(!)
722	(m)(2)	−722	(m)(")
723	(m)(3)	−723	(m)(#)
724	(m)(4)	−724	(m)($)
725	(m)(5)	−725	(m)(%)
726	(m)(6)	−726	(m)(&)
727	(m)(7)	−727	(m)(')
728	(m)(8)	−728	(m)("(")
729	(m)(9)	−729	(m)(")")
730	(m)(:)	−730	(m)(*)
731	(m)(;)	−731	(m)(+)
732	(m)(<)	−732	(m)(,)
733	(m)(=)	−733	(m)(-)
734	(m)(>)	−734	(m)(.)
735	(m)(?)	−735	(m)(/)
736	(n)(0)	−736	(n)(SP)
737	(n)(1)	−737	(n)(!)
738	(n)(2)	−738	(n)(")
739	(n)(3)	−739	(n)(#)
740	(n)(4)	−740	(n)($)
741	(n)(5)	−741	(n)(%)
742	(n)(6)	−742	(n)(&)
743	(n)(7)	−743	(n)(')
744	(n)(8)	−744	(n)("(")
745	(n)(9)	−745	(n)(")")
746	(n)(:)	−746	(n)(*)
747	(n)(;)	−747	(n)(+)
748	(n)(<)	−748	(n)(,)
749	(n)(=)	−749	(n)(-)

Table B-1, Continued

REPRESENTATION OF NUMBERS AS <INT> PARAMETERS

n	<int : n>	−n	<int : −n>	n	<int : n>	−n	<int : −n>	n	<int : n>	−n	<int : −n>
750	(n)(>)	−750	(n)(.)	800	(r)(0)	−800	(r)(SP)	850	(u)(2)	−850	(u)(")
751	(n)(?)	−751	(n)(/)	801	(r)(1)	−801	(r)(!)	851	(u)(3)	−851	(u)(#)
752	(o)(0)	−752	(o)(SP)	802	(r)(2)	−802	(r)(")	852	(u)(4)	−852	(u)($)
753	(o)(1)	−753	(o)(!)	803	(r)(3)	−803	(r)(#)	853	(u)(5)	−853	(u)(%)
754	(o)(2)	−754	(o)(")	804	(r)(4)	−804	(r)($)	854	(u)(6)	−854	(u)(&)
755	(o)(3)	−755	(o)(#)	805	(r)(5)	−805	(r)(%)	855	(u)(7)	−855	(u)(')
756	(o)(4)	−756	(o)($)	806	(r)(6)	−806	(r)(&)	856	(u)(8)	−856	(u)("(")
757	(o)(5)	−757	(o)(%)	807	(r)(7)	−807	(r)(')	857	(u)(9)	−857	(u)(")")
758	(o)(6)	−758	(o)(&)	808	(r)("(")	−808	(r)("(")	858	(u)(:)	−858	(u)(*)
759	(o)(7)	−759	(o)(')	809	(r)(9)	−809	(r)(")")	859	(u)(;)	−859	(u)(+)
760	(o)(8)	−760	(o)("(")	810	(r)(:)	−810	(r)(*)	860	(u)(<)	−860	(u)(,)
761	(o)(9)	−761	(o)(")")	811	(r)(;)	−811	(r)(+)	861	(u)(=)	−861	(u)(-)
762	(o)(:)	−762	(o)(*)	812	(r)(<)	−812	(r)(,)	862	(u)(>)	−862	(u)(.)
763	(o)(;)	−763	(o)(+)	813	(r)(=)	−813	(r)(-)	863	(u)(?)	−863	(u)(/)
764	(o)(<)	−764	(o)(,)	814	(r)(>)	−814	(r)(.)	864	(v)(0)	−864	(v)(SP)
765	(o)(=)	−765	(o)(-)	815	(r)(?)	−815	(r)(/)	865	(v)(1)	−865	(v)(!)
766	(o)(>)	−766	(o)(.)	816	(s)(0)	−816	(s)(SP)	866	(v)(2)	−866	(v)(")
767	(o)(?)	−767	(o)(/)	817	(s)(1)	−817	(s)(!)	867	(v)(3)	−867	(v)(#)
768	(p)(0)	−768	(p)(SP)	818	(s)(2)	−818	(s)(")	868	(v)(4)	−868	(v)($)
769	(p)(1)	−769	(p)(!)	819	(s)(3)	−819	(s)(#)	869	(v)(5)	−869	(v)(%)
770	(p)(2)	−770	(p)(")	820	(s)(4)	−820	(s)($)	870	(v)(6)	−870	(v)(&)
771	(p)(3)	−771	(p)(#)	821	(s)(5)	−821	(s)(%)	871	(v)(7)	−871	(v)(')
772	(p)(4)	−772	(p)($)	822	(s)(6)	−822	(s)(&)	872	(v)(8)	−872	(v)("(")
773	(p)(5)	−773	(p)(%)	823	(s)(7)	−823	(s)(')	873	(v)(9)	−873	(v)(")")
774	(p)(6)	−774	(p)(&)	824	(s)(8)	−824	(s)("(")	874	(v)(:)	−874	(v)(*)
775	(p)(7)	−775	(p)(')	825	(s)(9)	−825	(s)(")")	875	(v)(;)	−875	(v)(+)
776	(p)(8)	−776	(p)("(")	826	(s)(:)	−826	(s)(*)	876	(v)(<)	−876	(v)(,)
777	(p)(9)	−777	(p)(")")	827	(s)(;)	−827	(s)(+)	877	(v)(=)	−877	(v)(-)
778	(p)(:)	−778	(p)(*)	828	(s)(<)	−828	(s)(,)	878	(v)(>)	−878	(v)(.)
779	(p)(;)	−779	(p)(+)	829	(s)(=)	−829	(s)(-)	879	(v)(?)	−879	(v)(/)
780	(p)(<)	−780	(p)(,)	830	(s)(>)	−830	(s)(.)	880	(w)(0)	−880	(w)(SP)
781	(p)(=)	−781	(p)(-)	831	(s)(?)	−831	(s)(/)	881	(w)(1)	−881	(w)(!)
782	(p)(>)	−782	(p)(.)	832	(t)(0)	−832	(t)(SP)	882	(w)(2)	−882	(w)(")
783	(p)(?)	−783	(p)(/)	833	(t)(1)	−833	(t)(!)	883	(w)(3)	−883	(w)(#)
784	(q)(0)	−784	(q)(SP)	834	(t)(2)	−834	(t)(")	884	(w)(4)	−884	(w)($)
785	(q)(1)	−785	(q)(!)	835	(t)(3)	−835	(t)(#)	885	(w)(5)	−885	(w)(%)
786	(q)(2)	−786	(q)(")	836	(t)(4)	−836	(t)($)	886	(w)(6)	−886	(w)(&)
787	(q)(3)	−787	(q)(#)	837	(t)(5)	−837	(t)(%)	887	(w)(7)	−887	(w)(')
788	(q)(4)	−788	(q)($)	838	(t)(6)	−838	(t)(&)	888	(w)(8)	−888	(w)("(")
789	(q)(5)	−789	(q)(%)	839	(t)(7)	−839	(t)(')	889	(w)(9)	−889	(w)(")")
790	(q)(6)	−790	(q)(&)	840	(t)(8)	−840	(t)("(")	890	(w)(:)	−890	(w)(*)
791	(q)(7)	−791	(q)(')	841	(t)(9)	−841	(t)(")")	891	(w)(;)	−891	(w)(+)
792	(q)(8)	−792	(q)("(")	842	(t)(:)	−842	(t)(*)	892	(w)(<)	−892	(w)(,)
793	(q)(9)	−793	(q)(")")	843	(t)(;)	−843	(t)(+)	893	(w)(=)	−893	(w)(-)
794	(q)(:)	−794	(q)(*)	844	(t)(<)	−844	(t)(,)	894	(w)(>)	−894	(w)(.)
795	(q)(;)	−795	(q)(+)	845	(t)(=)	−845	(t)(-)	895	(w)(?)	−895	(w)(/)
796	(q)(<)	−796	(q)(,)	846	(t)(>)	−846	(t)(.)	896	(x)(0)	−896	(x)(SP)
797	(q)(=)	−797	(q)(-)	847	(t)(?)	−847	(t)(/)	897	(x)(1)	−897	(x)(!)
798	(q)(>)	−798	(q)(.)	848	(u)(0)	−848	(u)(SP)	898	(x)(2)	−898	(x)(")
799	(q)(?)	−799	(q)(/)	849	(u)(1)	−849	(u)(!)	899	(x)(3)	−899	(x)(#)

Table B-1, Continued

REPRESENTATION OF NUMBERS AS <INT> PARAMETERS

n	<int : n>	−n	<int : −n>
900	(x)(4)	−900	(x)($)
901	(x)(5)	−901	(x)(%)
902	(x)(6)	−902	(x)(&)
903	(x)(7)	−903	(x)(')
904	(x)(8)	−904	(x)("(")
905	(x)(9)	−905	(x)(")")
906	(x)(:)	−906	(x)(*)
907	(x)(;)	−907	(x)(+)
908	(x)(<)	−908	(x)(,)
909	(x)(=)	−909	(x)(-)
910	(x)(>)	−910	(x)(.)
911	(x)(?)	−911	(x)(/)
912	(y)(0)	−912	(y)(SP)
913	(y)(1)	−913	(y)(!)
914	(y)(2)	−914	(y)(")
915	(y)(3)	−915	(y)(#)
916	(y)(4)	−916	(y)($)
917	(y)(5)	−917	(y)(%)
918	(y)(6)	−918	(y)(&)
919	(y)(7)	−919	(y)(')
920	(y)(8)	−920	(y)("(")
921	(y)(9)	−921	(y)(")")
922	(y)(:)	−922	(y)(*)
923	(y)(;)	−923	(y)(+)
924	(y)(<)	−924	(y)(,)
925	(y)(=)	−925	(y)(-)
926	(y)(>)	−926	(y)(.)
927	(y)(?)	−927	(y)(/)
928	(z)(0)	−928	(z)(SP)
929	(z)(1)	−929	(z)(!)
930	(z)(2)	−930	(z)(")
931	(z)(3)	−931	(z)(#)
932	(z)(4)	−932	(z)($)
933	(z)(5)	−933	(z)(%)
934	(z)(6)	−934	(z)(&)
935	(z)(7)	−935	(z)(')
936	(z)(8)	−936	(z)("(")
937	(z)(9)	−937	(z)(")")
938	(z)(:)	−938	(z)(*)
939	(z)(;)	−939	(z)(+)
940	(z)(<)	−940	(z)(,)
941	(z)(=)	−941	(z)(-)
942	(z)(>)	−942	(z)(.)
943	(z)(?)	−943	(z)(/)
944	({)(0)	−944	({)(SP)
945	({)(1)	−945	({)(!)
946	({)(2)	−946	({)(")
947	({)(3)	−947	({)(#)
948	({)(4)	−948	({)($)
949	({)(5)	−949	({)(%)
950	({)(6)	−950	({)(&)
951	({)(7)	−951	({)(')
952	({)(8)	−952	({)("(")
953	({)(9)	−953	({)(")")
954	({)(:)	−954	({)(*)
955	({)(;)	−955	({)(+)
956	({)(<)	−956	({)(,)
957	({)(=)	−957	({)(-)
958	({)(>)	−958	({)(.)
959	({)(?)	−959	({)(/)
960	(|)(0)	−960	(|)(SP)
961	(|)(1)	−961	(|)(!)
962	(|)(2)	−962	(|)(")
963	(|)(3)	−963	(|)(#)
964	(|)(4)	−964	(|)($)
965	(|)(5)	−965	(|)(%)
966	(|)(6)	−966	(|)(&)
967	(|)(7)	−967	(|)(')
968	(|)(8)	−968	(|)("(")
969	(|)(9)	−969	(|)(")")
970	(|)(:)	−970	(|)(*)
971	(|)(;)	−971	(|)(+)
972	(|)(<)	−972	(|)(,)
973	(|)(=)	−973	(|)(-)
974	(|)(>)	−974	(|)(.)
975	(|)(?)	−975	(|)(/)
976	(})(0)	−976	(})(SP)
977	(})(1)	−977	(})(!)
978	(})(2)	−978	(})(")
979	(})(3)	−979	(})(#)
980	(})(4)	−980	(})($)
981	(})(5)	−981	(})(%)
982	(})(6)	−982	(})(&)
983	(})(7)	−983	(})(')
984	(})(8)	−984	(})("(")
985	(})(9)	−985	(})(")")
986	(})(:)	−986	(})(*)
987	(})(;)	−987	(})(+)
988	(})(<)	−988	(})(,)
989	(})(=)	−989	(})(-)
990	(})(>)	−990	(})(.)
991	(})(?)	−991	(})(/)
992	(~)(0)	−992	(~)(SP)
993	(~)(1)	−993	(~)(!)
994	(~)(2)	−994	(~)(")
995	(~)(3)	−995	(~)(#)
996	(~)(4)	−996	(~)($)
997	(~)(5)	−997	(~)(%)
998	(~)(6)	−998	(~)(&)
999	(~)(7)	−999	(~)(')
1000	(~)(8)	−1000	(~)("(")
1001	(~)(9)	−1001	(~)(")")
1002	(~)(:)	−1002	(~)(*)
1003	(~)(;)	−1003	(~)(+)
1004	(~)(<)	−1004	(~)(,)
1005	(~)(=)	−1005	(~)(-)
1006	(~)(>)	−1006	(~)(.)
1007	(~)(?)	−1007	(~)(/)
1008	(DEL)(0)	−1008	(DEL)(SP)
1009	(DEL)(1)	−1009	(DEL)(!)
1010	(DEL)(2)	−1010	(DEL)(")
1011	(DEL)(3)	−1011	(DEL)(#)
1012	(DEL)(4)	−1012	(DEL)($)
1013	(DEL)(5)	−1013	(DEL)(%)
1014	(DEL)(6)	−1014	(DEL)(&)
1015	(DEL)(7)	−1015	(DEL)(')
1016	(DEL)(8)	−1016	(DEL)("(")
1017	(DEL)(9)	−1017	(DEL)(")")
1018	(DEL)(:)	−1018	(DEL)(*)
1019	(DEL)(;)	−1019	(DEL)(+)
1020	(DEL)(<)	−1020	(DEL)(,)
1021	(DEL)(=)	−1021	(DEL)(-)
1022	(DEL)(>)	−1022	(DEL)(.)
1023	(DEL)(?)	−1023	(DEL)(/)
1024	(A)(@)(0)	−1024	(A)(@)(SP)
1025	(A)(@)(1)	−1025	(A)(@)(!)
1026	(A)(@)(2)	−1026	(A)(@)(")
1027	(A)(@)(3)	−1027	(A)(@)(#)
1028	(A)(@)(4)	−1028	(A)(@)($)
1029	(A)(@)(5)	−1029	(A)(@)(%)
1030	(A)(@)(6)	−1030	(A)(@)(&)
1031	(A)(@)(7)	−1031	(A)(@)(')
1032	(A)(@)(8)	−1032	(A)(@)("(")
1033	(A)(@)(9)	−1033	(A)(@)(")")
1034	(A)(@)(:)	−1034	(A)(@)(*)
1035	(A)(@)(;)	−1035	(A)(@)(+)
1036	(A)(@)(<)	−1036	(A)(@)(,)
1037	(A)(@)(=)	−1037	(A)(@)(-)
1038	(A)(@)(>)	−1038	(A)(@)(.)
1039	(A)(@)(?)	−1039	(A)(@)(/)
1040	(A)(A)(0)	−1040	(A)(A)(SP)
1041	(A)(A)(1)	−1041	(A)(A)(!)
1042	(A)(A)(2)	−1042	(A)(A)(")
1043	(A)(A)(3)	−1043	(A)(A)(#)
1044	(A)(A)(4)	−1044	(A)(A)($)
1045	(A)(A)(5)	−1045	(A)(A)(%)
1046	(A)(A)(6)	−1046	(A)(A)(&)
1047	(A)(A)(7)	−1047	(A)(A)(')
1048	(A)(A)(8)	−1048	(A)(A)("(")
1049	(A)(A)(9)	−1049	(A)(A)(")")

Appendix C

EXAMPLES OF CODE

This appendix contains two examples of host computer code used to control the 4113 terminal.

The first example is a PASCAL program which does graphic input using the thumbwheels and locator function. It includes PASCAL device driver routines to issue a variety of commands to the terminal.

The second example is a collection of FORTRAN subprograms which were used to test the terminal's block mode communications protocol.

A PASCAL GRAPHIC INPUT PROGRAM

```
13-May-81           TEK Pascal v3.00

   10   PROGRAM Example;
   20   {*************************************************************************
   30   *     This program shows how to do graphic input with the thumbwheels  *
   40   *  device and locator function.                                         *
   50   *     The program also shows how device driver routines might be        *
   60   *  written in PASCAL.                                                   *
   70   *     The PASCAL dialect used has the following peculiarities:          *
   80   *  (a) the native character set is ASCII; thus Chr(65) = 'A'.           *
   90   *  (b) the special file identifier TTY refers to the user's terminal.   *
  100   *************************************************************************}
  110
  120   CONST
  130      {** Mnemonic names for ASCII decimal equivalents **}
  140         Esc = 27; Us = 31; Gs = 29; Fs = 28;
  150         LetterA = 65; LetterB = 66; LetterC = 67; LetterD = 68;
  160         LetterE = 69; LetterF = 70; LetterG = 71; LetterH = 72;
  170         LetterI = 73; LetterJ = 74; LetterK = 75; LetterL = 76;
  180         LetterM = 77; LetterN = 78; LetterO = 79; LetterP = 80;
  190         LetterQ = 81; LetterR = 82; LetterS = 83; LetterT = 84;
  200         LetterU = 85; LetterV = 86; LetterW = 87; LetterX = 88;
  210         LetterY = 89; LetterZ = 90;
  220
  230   TYPE
  240      {** Data types used by I/O routines  **}
  250
  260         TwelveBitType = 0..4095;
  270         SevenBitType = 0..127;
  280         IntType = -32768..+32767;
  290         CharType = ' '..'~';
  300
  310         {** IntArrays are linked lists of IntRecords **}
  320            IntRecordPtrType = ^IntRecordType;
  330            IntRecordType = RECORD
  340               Item : IntType;
  350               Next : IntRecordPtrType;
  360               END;
  370            IntArrayType = RECORD
  380               Count : IntType;
  390               First : IntRecordPtrType;
  400               END;
  410
  420         XyType = RECORD
  430            X : TwelveBitType;
  440            Y : TwelveBitType;
  450            END;
  460
```

```
PAGE    2

VAR
   {** Variables used by I/O routines **}
      {** Used by SendXy **}
         OldHiY, OldExtra, OldLoY, OldHiX : 0..127;
         HaveSentXY : BOOLEAN;
      {** Used by ReceiveAscii **}
         EndOfLine : BOOLEAN;

   {** Variables used by the mainline. **}
      Xy : XyType;
      IntArray : IntArrayType;
      SignatureChar, KeyChar : CHAR;


{** Data structure manipulation routines : IntArrayType variables **}

PROCEDURE InitializeNewIntArray(VAR IntArray : IntArrayType); {}{}{}{}{}
{} BEGIN                                                               {}
{} IntArray.Count := 0;                                                {}
{} IntArray.First := NIL                                               {}
{} END;                                                                {}
{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}

PROCEDURE ClearIntArray(VAR IntArray : IntArrayType); {}{}{}{}{}{}{}{}{}
{} VAR                                                                 {}
{}    Pointer : IntRecordPtrType;                                      {}
{} BEGIN                                                               {}
{} IF (IntArray.Count > 0)                                             {}
{}    THEN BEGIN                                                       {}
{}    IF (IntArray.First^.Next = NIL)                                  {}
{}       THEN BEGIN                                                    {}
{}       Dispose(IntArray.First);                                      {}
{}       IntArray.First := NIL;                                        {}
{}       IntArray.Count := 0                                           {}
{}       END                                                           {}
{}    ELSE                                                             {}
{}       BEGIN                                                         {}
{}       Pointer := IntArray.First;                                    {}
{}       WHILE NOT (Pointer^.Next^.Next = NIL)                         {}
{}          DO Pointer := Pointer^.Next;                               {}
{}       Dispose(Pointer^.Next);                                       {}
{}       Pointer^.Next := NIL;                                         {}
{}       IntArray.Count := IntArray.Count - 1;                         {}
{}       ClearIntArray(IntArray)                                       {}
{}       END                                                           {}
{}    END;                                                             {}
{} END;                                                                {}
{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}

PROCEDURE GetNewIntRecordAndLoadIt(VAR Pointer : IntRecordPtrType;
{}                                     Int : IntType);{}{}{}{}{}{}{}{}{}{}{}
{} BEGIN                                                               {}
{} New(Pointer);                                                       {}
{} Pointer^.Item := Int;                                               {}
{} Pointer^.Next := NIL                                                {}
{} END;                                                                {}
{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}
```

```
PROCEDURE AppendToIntArray(VAR IntArray : IntArrayType;
{}                        Int : IntType);{}{}{}{}{}{}{}{}{}{}{}{}{}{}
{} VAR                                                              {}
{}    Pointer : IntRecordPtrType;                                   {}
{}                                                                  {}
{} BEGIN                                                            {}
{} IF (IntArray.Count = 0)                                          {}
{}    THEN BEGIN                                                    {}
{}    GetNewIntRecordAndLoadIt(IntArray.First,Int);                 {}
{}    IntArray.Count := IntArray.Count + 1                          {}
{}    END                                                           {}
{} ELSE IF (IntArray.Count > 0)                                     {}
{}    THEN BEGIN                                                    {}
{}    Pointer := IntArray.First;                                    {}
{}    WHILE NOT (Pointer^.Next = NIL)                               {}
{}       DO Pointer := Pointer^.Next;                               {}
{}    GetNewIntRecordAndLoadIt(Pointer^.Next,Int);                  {}
{}    IntArray.Count := IntArray.Count + 1                          {}
{}    END;                                                          {}
{} END;                                                             {}
{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}



{** Routines for output from host to 411X terminal **}

PROCEDURE SendASCII(CharCode : SevenBitType); {}{}{}{}{}{}{}{}{}{}{}{}
{} BEGIN                                                            {}
{} Write(TTY,Chr(CharCode));                                        {}
{} END;                                                             {}
{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}

PROCEDURE SendXY(XY : XYType);{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}
{}                                                                  {}
{} TYPE                                                             {}
{}    FiveBitType = 0..31;                                          {}
{}    TwoBitType = 0..3;                                            {}
{}                                                                  {}
{} VAR                                                              {}
{}    HiFiveX, HiFiveY, MidFiveX, MidFiveY : FiveBitType;           {}
{}    LoTwoX, LoTwoY : TwoBitType;                                  {}
{}    HiY, Extra, LoY, HiX, LoX : 0..127;                           {}
{}    GottaSendExtraByte, GottaSendLoYByte,                         {}
{}    GottaSendHiXByte : BOOLEAN;                                   {}
{}                                                                  {}
{} PROCEDURE Split(Coordinate : TwelveBitType;                      {}
{} {}       VAR HighFiveBits : FiveBitType;                         {}
{} {}       VAR MiddleFiveBits : FiveBitType;                       {}
{} {}       VAR LowTwoBits : TwoBitType); {}{}{}{}{}{}{}{}{}{}{}{} {}
{} {}                                                           {} {}
{} {} BEGIN                                                     {} {}
{} {} HighFiveBits := Coordinate DIV 128;                       {} {}
{} {} MiddleFiveBits := (Coordinate MOD 128) DIV 4;             {} {}
{} {} LowTwoBits := Coordinate MOD 4;                           {} {}
{} {} END;                                                      {} {}
{} {}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{} {}
{}                                                                  {}
```

PAGE 3

```
{} BEGIN                                                                    {}
{} {** Partition the X- and Y- coordinates into 5-, 5-,2-bit fields*}     {}
{}    Split(XY.X, HiFiveX, MidFiveX, LoTwoX);                              {}
{}    Split(XY.Y, HiFiveY, MidFiveY, LoTwoY);                              {}
{} {** Compute all five bytes of the <xy> paramter **}                    {}
{}    HiY := HiFiveY + 32;                                                  {}
{}    Extra := 4 * LoTwoY + LoTwoX + 96;                                    {}
{}    LoY := MidFiveY + 96;                                                 {}
{}    HiX := HiFiveX + 32;                                                  {}
{}    LoX := MidFiveX + 64;                                                 {}
{} {** Send only those bytes which are needed **}                          {}
{}    IF NOT HaveSentXY                                                     {}
{}       THEN                                                               {}
{}          BEGIN                                                           {}
{}          SendASCII(HiY); OldHiY := HiY;                                  {}
{}          SendASCII(Extra); OldExtra := Extra;                            {}
{}          SendASCII(LoY); OldLoY := LoY;                                  {}
{}          SendASCII(HiX); OldHiX := HiX;                                  {}
{}          SendASCII(LoX);                                                 {}
{}          HaveSentXY := TRUE;                                             {}
{}          END                                                             {}
{}       ELSE {** IF HaveSentXY **}                                         {}
{}          BEGIN                                                           {}
{}          IF (HiY <> OldHiY)                                              {}
{}             THEN BEGIN                                                   {}
{}             SendASCII(HiY); OldHiY := HiY;                               {}
{}             END;                                                         {}
{}          GottaSendExtraByte := (Extra <> OldExtra);                      {}
{}          GottaSendHiXByte := (HiX <> OldHiX);                            {}
{}          GottaSendLoYByte := (LoY <> OldLoY)                             {}
{}                                 OR GottaSendExtraByte                    {}
{}                                 OR GottaSendHiXByte;                     {}
{}          IF GottaSendExtraByte                                           {}
{}             THEN BEGIN                                                   {}
{}             SendASCII(Extra); OldExtra := Extra;                         {}
{}             END;                                                         {}
{}          IF GottaSendLoYByte                                             {}
{}             THEN BEGIN                                                   {}
{}             SendASCII(LoY); OldLoY := LoY;                               {}
{}             END;                                                         {}
{}          IF GottaSendHiXByte                                             {}
{}             THEN BEGIN                                                   {}
{}             SendASCII(HiX); OldHiX := HiX;                               {}
{}             END;                                                         {}
{}          SendASCII(LoX);                                                 {}
{}          END;                                                            {}
{} END;                                                                     {}
{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}
```

```
PAGE    4

PROCEDURE SendInt(I : IntType);
{*********************************************************************}
{*     Encodes I as a sequence of bytes in the <packed-integer>      *}
{*  format and sends those bytes to the terminal.                     *}
{*     Calls the SendASCII procedure; also uses the following         *}
{*  globally-declared data types:                                     *}
{*                                                                    *}
{*    TYPE                                                            *}
{*         SevenBitType = 0..127;                                     *}
{*                                                                    *}
{*********************************************************************}
{}                                                                   {}
{}    CONST                                                          {}
{}        MaxNumberBytes = 3; {*** I should be in the range from  }  {}
{}                            { -32768 to +32767, so no more than }  {}
{}                            { three bytes are needed.        ***}  {}
{}    VAR                                                            {}
{}        StackPointer, J : 0..MaxNumberBytes;                       {}
{}        Stack : ARRAY[1..MaxNumberBytes] OF SevenBitType;          {}
{}        HiI : 64..127;                                             {}
{}        LoI : 32..63;                                              {}
{}        Negative : BOOLEAN;                                        {}
{}                                                                   {}
{}    PROCEDURE Push(Byte : SevenBitType); {** Push byte on stack **} {}
{}    {}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}   {}
{}    {}  BEGIN                                                 {}   {}
{}    {}  StackPointer := StackPointer + 1;                     {}   {}
{}    {}  Stack[StackPointer] := Byte                           {}   {}
{}    {}  END;                                                  {}   {}
{}    {}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}   {}
{}                                                                   {}
```

```
PAGE    5

{}                                                                      {}
{}    BEGIN      {*** statement-part of SendInt procedure ***}          {}
{}                                                                      {}
{}    {*** Initialize things. ***}                                      {}
{}        StackPointer := 0;                                            {}
{}        Negative := FALSE;                                            {}
{}                                                                      {}
{}    {*** Compute bytes, push them onto the stack. ***}                {}
{}        IF I < 0                                                      {}
{}            THEN BEGIN                                                {}
{}            I := -I;                                                  {}
{}            Negative := TRUE;                                         {}
{}            END;                                                      {}
{}                                                                      {}
{}        {*** Compute LoI byte, push it on the stack. ***}             {}
{}        {*** <LoI> : binary 01sdddd; s = sign bit, d = data bit.***}{}
{}        {*** s=1 for positive numbers, s=0 for negative numbers. **}{}
{}            LoI := I MOD 16 + 32;                                     {}
{}            IF NOT Negative THEN LoI := LoI + 16;                     {}
{}            Push(LoI);                                                {}
{}            I := I DIV 16;                                            {}
{}                                                                      {}
{}        {*** Compute HiI bytes, push them on stack. ***}              {}
{}        {*** <HiI> : binary 1ddddd. ***}                              {}
{}            WHILE I > 0                                               {}
{}                DO BEGIN                                              {}
{}                HiI := (I MOD 64) + 64;                               {}
{}                Push(HiI);                                            {}
{}                I := I DIV 64                                         {}
{}                END;                                                  {}
{}                                                                      {}
{}    {*** Pop bytes off stack and send them to the terminal. ***}      {}
{}        FOR J := StackPointer DOWNTO 1                                {}
{}            DO SendASCII(Stack[J])                                    {}
{}                                                                      {}
{}    END; {*** of SendInt procedure ***}                               {}
{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}
```

PAGE 6

```
PROCEDURE SendIntArray(IntArray : IntArrayType);{}{}{}{}{}{}{}{}{}{}{}{}{}
{}                                                                      {}
{} VAR                                                                  {}
{}     I : IntType;                                                     {}
{}     Pointer : IntRecordPtrType;                                      {}
{}                                                                      {}
{} BEGIN                                                                {}
{} SendInt(IntArray.Count);                                             {}
{} IF (IntArray.Count > 0) THEN                                         {}
{}     BEGIN {** Find the <int>s in the linked list, send 'em off.**}   {}
{}     Pointer := IntArray.First;                                       {}
{}     FOR I := 1 TO IntArray.Count DO                                  {}
{}         BEGIN                                                        {}
{}         SendInt(Pointer^.Item);                                      {}
{}         Pointer := Pointer^.Next                                     {}
{}         END;                                                         {}
{}     END;                                                             {}
{} END;                                                                 {}
{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}
{** 411X I/O routines ... for input to host from 411X terminal **}

PROCEDURE ReceiveAscii(VAR CharacterCode : SevenBitType); {}{}{}{}{}{}{}
{}                                                                      {}
{}                                                                      {}
{}                                                                      {}
{} VAR                                                                  {}
{}     Ch : CHAR;                                                       {}
{}                                                                      {}
{} BEGIN                                                                {}
{} IF Eoln(TTY)                                                         {}
{}     THEN                                                             {}
{}         BEGIN                                                        {}
{}         CharacterCode := 13; {** (CR) **}                            {}
{}         Readln(TTY)                                                  {}
{}         END                                                          {}
{}     ELSE {** IF NOT Eoln(TTY) **}                                    {}
{}         BEGIN                                                        {}
{}         Read(TTY,Ch);                                                {}
{}         CharacterCode := Ord(Ch)                                     {}
{}         END;                                                         {}
{} END;                                                                 {}
{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}
```

PAGE 7

```
PROCEDURE ParseAnXyReport(VAR Xy : XyType); {}{}{}{}{}{}{}{}{}{}{}{}{}{}{}
{}                                                                      {}
{} VAR                                                                  {}
{}     HiYReport,ExtraReport,LoYReport,HiXReport,                       {}
{}       LoXReport : SevenBitType;                                      {}
{}     X, Y : TwelveBitType;                                            {}
{}                                                                      {}
{} PROCEDURE SkipPastIrrelevantCharacters(VAR Ch : SevenBitType);{}{}  {}
{} {} BEGIN                                                         {}  {}
{} {} ReceiveAscii(Ch);                                             {}  {}
{} {} WHILE ((Ch < 32) OR (Ch > 63) ) DO                            {}  {}
{} {}     ReceiveAscii(Ch);                                         {}  {}
{} {} END;                                                          {}  {}
{} {}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{} {}
{}                                                                      {}
{} BEGIN                                                                {}
{} {** grab the five characters **}                                     {}
{}     SkipPastIrrelevantCharacters(HiYReport);                         {}
{}     SkipPastIrrelevantCharacters(ExtraReport);                       {}
{}     SkipPastIrrelevantCharacters(LoYReport);                         {}
{}     SkipPastIrrelevantCharacters(HiXReport);                         {}
{}     SkipPastIrrelevantCharacters(LoXReport);                         {}
{} {** Compute the x- and y-coordinates **}                             {}
{}     X := 32*(HiXReport - 32) + (LoXReport - 32);                     {}
{}     X := 4*X + (ExtraReport MOD 4);                                  {}
{}     Y := 32*(HiYReport - 32) + (LoYReport - 32);                     {}
{}     Y := 4*Y + (ExtraReport DIV 4) MOD 4;                            {}
{}     Xy.X := X;                                                       {}
{}     Xy.Y := Y;                                                       {}
{} END;                                                                 {}
{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}

PROCEDURE ParseAGinLocatorReport(VAR Key : CHAR;
{}                                 VAR Xy : XyType);{}{}{}{}{}{}{}{}{}{}{}
{}                                                                      {}
{} VAR                                                                  {}
{}     AsciiDecimalEquivalent : SevenBitType;                           {}
{}                                                                      {}
{} BEGIN                                                                {}
{} {** Parse the <ASCII-char : "key"> **}                               {}
{}     ReceiveASCII(AsciiDecimalEquivalent);                            {}
{}     Key := Chr(AsciiDecimalEquivalent);                              {}
{} {** Parse the <xy-report : "cursor-position">                        {}
{}     ParseAnXyReport(Xy)                                              {}
{} END;                                                                 {}
{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}
```

PAGE 8

```
{** Routines to issue commands to the terminal **}

PROCEDURE BeginSegment(SegmentNumber : IntType);{}{}{}{}{}{}{}{}{}{}{}{}
{} BEGIN                                                               {}
{} {** (ESC)(S)(O) **}                                                 {}
{}     SendASCII(Esc);                                                 {}
{}     SendASCII(LetterS);                                             {}
{}     SendASCII(LetterO);                                             {}
{} SendInt(SegmentNumber)                                              {}
{} END;                                                                {}
{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}
                                                                         
PROCEDURE DeleteSegment(SegmentNumber : IntType); {}{}{}{}{}{}{}{}{}{}{}
{} BEGIN                                                               {}
{} {** (ESC)(S)(K) **}                                                 {}
{}     SendASCII(Esc);                                                 {}
{}     SendASCII(LetterS);                                             {}
{}     SendASCII(LetterK);                                             {}
{} SendInt(SegmentNumber);                                             {}
{} END;                                                                {}
{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}

PROCEDURE DisableGin(DeviceAndFunction : IntType);{}{}{}{}{}{}{}{}{}{}{}
{} BEGIN                                                               {}
{} {** (ESC)(I)(D) **}                                                 {}
{}     SendASCII(Esc);                                                 {}
{}     SendASCII(LetterI);                                             {}
{}     SendASCII(LetterD);                                             {}
{} SendInt(DeviceAndFunction)                                          {}
{} END;                                                                {}
{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}

PROCEDURE Draw(Xy : XyType);{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}
{} BEGIN                                                               {}
{} {** (ESC)(L)(G) **}                                                 {}
{}     SendASCII(Esc);                                                 {}
{}     SendASCII(LetterL);                                             {}
{}     SendASCII(LetterG);                                             {}
{} SendXy(Xy)                                                          {}
{} END;                                                                {}
{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}
```

PAGE 9

```
PROCEDURE EnableDialogArea(EnableMode : IntType); {}{}{}{}{}{}{}{}{}{}{}{}
{} BEGIN                                                                {}
{} {** (ESC)(K)(A) **}                                                  {}
{}    SendASCII(Esc);                                                   {}
{}    SendASCII(LetterK);                                               {}
{}    SendASCII(LetterA);                                               {}
{} SendInt(EnableMode)                                                  {}
{} END;                                                                 {}
{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}

PROCEDURE EnableGin(DeviceAndFunction,
{}                   NumberOfEvents : IntType);{}{}{}{}{}{}{}{}{}{}{}{}{}{}
{} BEGIN                                                                {}
{} {** (ESC)(I)(E) **}                                                  {}
{}    SendASCII(Esc);                                                   {}
{}    SendASCII(LetterI);                                               {}
{}    SendASCII(LetterE);                                               {}
{} SendInt(DeviceAndFunction);                                          {}
{} SendInt(NumberOfEvents);                                             {}
{} END;                                                                 {}
{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}

PROCEDURE EndSegment; {}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}
{} BEGIN                                                                {}
{} {** (ESC)(S)(C) **}                                                  {}
{}    SendASCII(Esc);                                                   {}
{}    SendASCII(LetterS);                                               {}
{}    SendASCII(LetterC);                                               {}
{} END;                                                                 {}
{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}

PROCEDURE Move(Xy : XyType);{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}
{} BEGIN                                                                {}
{} {** (ESC)(L)(F) **}                                                  {}
{}    SendASCII(Esc);                                                   {}
{}    SendASCII(LetterL);                                               {}
{}    SendASCII(LetterF);                                               {}
{} SendXy(Xy);                                                          {}
{} END;                                                                 {}
{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}

PROCEDURE SetDialogAreaVisibility(VisibilityMode : IntType);{}{}{}{}{}{}
{} BEGIN                                                                {}
{} {** (ESC)(L)(V) **}                                                  {}
{}    SendASCII(Esc);                                                   {}
{}    SendASCII(LetterL);                                               {}
{}    SendASCII(LetterV);                                               {}
{} SendInt(VisibilityMode)                                              {}
{} END;                                                                 {}
{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}
```

PAGE 10

```
PROCEDURE SetEolString(EolString : IntArrayType); {}{}{}{}{}{}{}{}{}{}{}{}
{} BEGIN                                                                 {}
{} {** (ESC)(N)(T) **}                                                   {}
{}     SendASCII(Esc);                                                   {}
{}     SendASCII(LetterN);                                               {}
{}     SendASCII(LetterT);                                               {}
{} SendIntArray(EolString);                                              {}
{} END;                                                                  {}
{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}

PROCEDURE SetGinGridding(DeviceAndFunction : IntType;
{}                         XGridSpacing, YGridSpacing : IntType); {}{}{}{}
{} BEGIN                                                                 {}
{} {** (ESC)(I)(G) **}                                                   {}
{}     SendASCII(Esc);                                                   {}
{}     SendASCII(LetterI);                                               {}
{}     SendASCII(LetterG);                                               {}
{} SendInt(DeviceAndFunction);                                           {}
{} SendInt(XGridSpacing);                                                {}
{} SendInt(YGridSpacing);                                                {}
{} END;                                                                  {}
{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}

PROCEDURE SetGinRubberbanding(DeviceAndFunction : IntType;
{}                              Rubberbanding : IntType); {}{}{}{}{}{}{}{}
{} BEGIN                                                                 {}
{} {** (ESC)(I)(R) **}                                                   {}
{}     SendASCII(Esc);                                                   {}
{}     SendASCII(LetterI);                                               {}
{}     SendASCII(LetterR);                                               {}
{} SendInt(DeviceAndFunction);                                           {}
{} Sendint(Rubberbanding);                                               {}
{} END;                                                                  {}
{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}

PROCEDURE SetReportEomFrequency(FrequencyCode : IntType); {}{}{}{}{}{}{}{}
{} BEGIN                                                                 {}
{} {** (ESC)(I)(L) **}                                                   {}
{}     SendASCII(Esc);                                                   {}
{}     SendASCII(LetterI);                                               {}
{}     SendASCII(LetterM);                                               {}
{} SendInt(FrequencyCode);                                               {}
{} END;                                                                  {}
{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}
```

PAGE 11

```
PROCEDURE SetReportMaxLineLength(Length : IntType); {}{}{}{}{}{}{}{}{}{}{}
{} BEGIN                                                                 {}
{} {** (ESC)(I)(M) **}                                                   {}
{}    SendASCII(Esc);                                                    {}
{}    SendASCII(LetterI);                                                {}
{}    SendASCII(LetterL);                                                {}
{} SendInt(Length); {** a length of zero disables the feature **}        {}
{} END;                                                                  {}
{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}

PROCEDURE SetReportSigChars(ReportTypeCode,
{}                          SigChar,
{}                          TermSigChar : IntType); {}{}{}{}{}{}{}{}{}{}{}
{} BEGIN                                                                 {}
{} {** (ESC)(I)(S) **}                                                   {}
{}    SendASCII(Esc);                                                    {}
{}    SendASCII(LetterI);                                                {}
{}    SendASCII(LetterS);                                                {}
{} SendInt(ReportTypeCode);                                              {}
{} SendInt(SigChar);                                                     {}
{} SendInt(TermSigChar);                                                 {}
{} END;                                                                  {}
{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}

{** Routines used by the mainline **}

PROCEDURE Handshake;{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}
{} VAR                                                                   {}
{}    I : 1..5;                                                          {}
{}                                                                       {}
{} BEGIN                                                                 {}
{} {** Issue <report-4010-status> command **}                            {}
{}    SendASCII(Esc);                                                    {}
{}    SendASCII(5); {** (ENQ) **}                                        {}
{} {** Parse the <4010-status-report>, crudely **}                       {}
{}    IF Eoln(TTY) THEN Readln(TTY);                                     {}
{}    FOR I := 1 TO 5 DO Get(TTY);                                       {}
{} END;                                                                  {}
{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}

PROCEDURE DisplayInstructions;{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}
{} BEGIN                                                                 {}
{} Writeln(TTY);                                                         {}
{} Writeln(TTY,'Type - ');                                               {}
{} Writeln(TTY,'  M to move,');                                          {}
{} Writeln(TTY,'  D to draw,');                                          {}
{} Writeln(TTY,'  X to exit this program.');                             {}
{} END;                                                                  {}
{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}
```

PAGE 12

```
PROCEDURE ParseGinReportItemAndTakeAppropriateAction; {}{}{}{}{}{}{}{}{}{}
{} BEGIN                                                                  {}
{} {** Look for signature character **}                                   {}
{}    WHILE NOT (TTY^ IN ['W', 'w' ]){* while next char is not W *}       {}
{}       DO BEGIN {** skip past the non-W character **}                   {}
{}          IF Eoln(TTY) {** If next character is a (CR) **}              {}
{}             THEN                                                       {}
{}                Readln(TTY) {** advance to 1st char of next line **}{}
{}             ELSE                                                       {}
{}                Get(TTY) {** advance to next char **}                   {}
{}       END;                                                             {}
{} Read(TTY,SignatureChar);                                               {}
{} IF (SignatureChar = 'W')                                               {}
{}    THEN {** Parse a <GIN-locator-report>. **}                          {}
{}       BEGIN                                                            {}
{}       Read(TTY,KeyChar);                                               {}
{}       ParseAnXyReport(Xy);                                             {}
{}       IF KeyChar IN ['M','m','D','d','X','x']                          {}
{}          THEN {** take the appropriate action **}                      {}
{}             CASE KeyChar OF                                            {}
{}             {} 'M', 'm' : Move(Xy);                                    {}
{}             {} 'D', 'd' : Draw(Xy);                                    {}
{}             {} 'X', 'x' : DisableGin(0);                               {}
{}             {} END {** of CASE statement **}                           {}
{}          ELSE  {** If he typed an invalid key char **}                 {}
{}             DisplayInstructions;                                       {}
{}       END;                                                             {}
{} END; {* of ParseGinReportItemAndTakeAppropriateAction procedure *} {}
{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}{}
```

```
PAGE  13

 5840   BEGIN
 5850   {** Prepare dialog area **}
 5860      EnableDialogArea(1);
 5870      SetDialogAreaVisibility(1);
 5880   {** Prepare for GIN **}
 5890      {** <set-EOL-string : (13)> **}
 5900         InitializeNewIntArray(IntArray);
 5910         AppendToIntArray(IntArray,13);
 5920         SetEolString(IntArray);
 5930      SetReportSigChars(0,87,119); {* (W), (w) *}
 5940      SetReportEomFrequency(1);    {* "more frequent " *}
 5950      SetReportMaxLineLength(72);
 5960      SetGinGridding(0,100,100);   {* enable gridding **}
 5970      SetGinRubberbanding(0,1);    {* enable rubberbanding *}
 5980   {** Tell operator what to do **}
 5990      Handshake; {* Be sure previous commands have been executed *}
 6000      DisplayInstructions;
 6010   {** Open a segment **}
 6020      DeleteSegment(-1); {** Delete all old segments. **}
 6030      BeginSegment(1);
 6040   {** Enable for GIN **}
 6050      EnableGin(0,32767); {** "32767" means "many points" **}
 6060   {** Loop until a <term-sig-char> is received. **}
 6070      REPEAT
 6080         ParseGinReportItemAndTakeAppropriateAction
 6090      UNTIL SignatureChar = 'w';
 6100   {** Close the segment. **}
 6110      EndSegment;
 6120
 6130
 6140   END.

0 Error messages issued.
0 Warning messages issued.

Highseg:    3P
Lowseg :    1P
```

FORTRAN BLOCK MODE COMMUNICATIONS DRIVERS

```
C INTRODUCTION.
C
C   This example I/O system for the TEKTRONIX 411X Computer Display
C   Terminals was implemented during the development of these terminals
C   and is presented as one way to do it.  The FORTRAN is very close to
C   the 1966 ANSI FORTRAN standard,  deviating in ways acceptable to
C   most modern compilers.  This I/O system supports not only the new
C   411X block mode but also the 466X Interactive Digital Plotter block
C   mode as well, allowing direct access to plotter features through
C   the use of the 411X <PORT-COPY> command.  This system assumes the
C   presence of two I/O routines from PLOT 10 Terminal Control System,
C   ADEIN and ADEOUT.  In fact, this system may be integrated with TCS
C   to provide access to the new terminals via block mode.
C
C USER INTERFACE.
C
C   These routines from this I/O system are the ones the user must
C   call to drive the system.
C
C       IOINIT  initializes the I/O system and the terminal.
C       IOEND   shuts down the I/O system.
C       KYBDIN  gets input from the keyboard, out of block mode.
C       FILIN   gets input from a terminal file during a <COPY> transfer.
C       SNDEOF  sends an end-of-file to the terminal to end a <COPY>.
C       BLOKGO  starts block mode transmissions.
C       BLKEND  ends block mode transmissions.
C       PLTRBM  turns plotter block mode on and off.
C       EMPTIN  empties the input buffer of any extraneous characters.
C       STIN    is the primary input routine.
C       DUMP    flushes the output buffer.
C       STOUT   is the primary output routine.
C       INTRAY  translates an integer array into terminal format.
C       ADERAY  translates an ADE array into terminal format.
C       INTPAK  translates an integer into terminal format.
C       RELPAK  translates a real into terminal format.
C       XYTRN   translates an XY pair into terminal format.
C       INTUNP  translates a terminal int-report into an integer.
C       RELUNP  trnaslates a terminal real-report into a real.
C       XYUNP   translates a terminal XY-report into an XY pair.
C       KIN2AS  translates an integer into a string of digits.
C       KAS2IN  translates a string of digits into an integer.
```

```
C   In general, IOINIT is called once at the beginning of a program.
C   IOEND is likewise called once at the end.  KYBDIN is useful during
C   block mode to allow the operator to use the normal system input
C   features to edit the input.  FILIN and SNDEOF are of use during
C   <COPY> operations between the host computer and the terminal.  BLOKGO
C   must be called to put the software into block mode, and BLKEND may
C   be called to exit block mode.  PLTRBM may be used to enable or
C   disable plotter block mode.  Plotter block mode is of use either
C   when the plotter is connected between the terminal and the host
C   computer or, if connected to a peripheral port, between a <PORT-COPY>
C   command and an end-of-file indicator.  This system assumes the plotter
C   address is 'A', and it sends plotter on and plotter off commands on
C   entering and leaving plotter block mode.  EMPTIN is used to make
C   sure no unwanted characters are in the system input buffer before
C   a call to STIN (or KYBDIN or FILIN).  STIN is the input 'hole'
C   through which all input flows.  DUMP is used to force STOUT to flush
C   the system output buffer, and should be called seldomly if ever by
C   the typical program.  STOUT is the output 'hole' through which all
C   output flows, taking care of plotter block mode if enabled.  Terminal
C   block mode is handled by lower level routines called by STIN and
C   STOUT and is not directly accessible to the user.  The last five
C   routines on the above list are very useful in formatting arguments
C   in commands to the terminal, but only XYTRN is of any use with the
C   plotter.
C
C INSTALLING THIS SYSTEM.
C
C   In implementing this system, you may wish to make the buffers and
C   line lengths smaller.  To do this, change the size of the arrays in
C   /COMM/ common to match your needs, change the initial values of
C   KBUFSZ in IOINIT and BLKEND and the values of JBLINE and JLINLN in
C   IOINIT.  The dimension of the arrays in BLKOUT and BLKIN may also
C   be made smaller.  Other installation considerations are discussed
C   in the following sections.
C
C ADEIN and ADEOUT.
C
C   These routines are part the PLOT 10 Terminal Control System, but if
C   you do not have them, here is a brief description of them.  ADEOUT
C   outputs an array of ASCII-decimal-equivalent (ADE) characters to
C   the terminal.  It should be able to transmit all 128 of the ASCII
C   characters, although only (ESC) and from (SP) to (_) are required
C   for this I/O system, if block mode with six-bit packing is used.
C   ADEIN inputs a line of characters from the terminal, ending with
C   but not including a carriage return.  Its first argument is the
C   number of characters in the line, not counting trailing spaces, and
C   its second argument is the array for the characters.
C
```

```
C INTEGRATING WITH TCS.
C
C   To use this I/O with the PLOT 10 Terminal Control System, replace
C   TOUTST with STOUT and have TINSTR simply call STIN.  In addition,
C   have INITT call IOINIT and FINITT call IOEND.  If you had to use
C   buffer type 1 or 2, use block mode now by calling BLOKGO from
C   INITT.  If you have 4662A01 or 4663A01 PLOT 10 Utility Routines,
C   integrate calls to PLTRBM into PLON and PLOFF, with calls to do a
C   <PORT-COPY> and to SNDEOF in PLINIT, PLON and PLOFF as needed.
C   ANMODE should call BLKEND to dump the buffer in block mode, but
C   should contain a copy of /COMM/ common and set the variable KBMODE
C   back to one so that the software will reenter block immediately.
C   XYCNVT would benefit from a replacement of much of its code with a
C   call to XYTRN, which was modelled after XYCNVT anyway.  ANSTR should
C   have the variable MAXLEN removed, including both executable lines of
C   code containing it.
C
C TRIMMING THIS SYSTEM.
C
C   Since this is just an example of an I/O system, any portion of it
C   may be trimmed out as desired.  The plotter block mode part of STOUT
C   is fairly easy to be rid of if attention is paid to the statement
C   numbers.  The terminal block mode packing and unpacking routines
C   may be reduced in size by removing one or the other of the two methods
C   provided in each, the 'easy way' and the 'hard way'.  Which part to
C   remove depends on the packing factors being used.  See the first
C   executable line of PACKER and UNPACK for clarification of this.  It
C   is also possible to remove many or all of the calls to the parameter
C   setting routines in IOINIT if the terminals being accessed are
C   already appropriately set up.  However, the variables in /COMM/ must
C   still be set properly so the software can do its job.
C
```

```
C THE CODE OF THE EXAMPLE I/O SYSTEM:
C
C
C----------SUBROUTINE--IOINIT---
C
      SUBROUTINE IOINIT
C * IOINIT INITIALIZES ALL BLOCK MODE PARAMETERS AND ALL STANDARD
C * COMM PARAMETERS TO SET UP THE SOFTWARE AND TERMINAL
C * THESE VALUES SHOULD WORK ON MOST SYSTEMS CAPABLE OF FULL ASCII
C * INPUT AND OUTPUT - LARGER LINE, BLOCK AND BUFFER SIZES MAY BE
C * DESIRABLE FOR SYSTEMS ABLE TO HANDLE THEM
C * SOME STANDARD COMM VALUES MAY NEED ADJUSTMENT FOR YOUR SYSTEM
      COMMON /COMRAY/ JCODE(256)
      COMMON /COMM/ KOUTPT,KOUTBF(512),KINEND,KINPT,KINBUF(512),
     & KRESLU,KIGDEL,KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),
     & KEOMC1,KEOMC2,KCHARS(4),KBUFSZ,KBPLOT,KBPSW,
     & KBARM,KBMODE,KHLENH,KHEADH(10),KHLENT,KHEADT(10),KCONTH,
     & KCONTT,KENDH,KENDT,KNXNOH,KNXMTH(20),KNXNOT,KNXMTT(20),
     & KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,KPACKT,KBLENH,KBLENT,
     & KBLINE,KBLOKH,KEOPH,KEOMH,KEOFH,KEOMT,KEOFT
      DIMENSION JPSTRG(2),JEOFST(1),JEOLST(1)
      DIMENSION JHEADT(5),JHEADH(5),JNXMTT(1),JNXMTH(6)
C * BLOCK MODE PARAMETER DATA STATEMENTS
C *         TERMINAL-TO-HOST          HOST-TO-TERMINAL
      DATA JHEADT /33,34,35,36,37/,JHEADH /38,125,125,41,42/
      DATA JNXMTT /126/,            JNXMTH /96,97,98,17,19,0/
      DATA JMASTT /126/,            JMASTH /96/
      DATA JCONTT /97/,             JCONTH /97/
      DATA JENDT  /98/,             JENDH  /98/
      DATA JBYTET /7/,              JBYTEH /7/
      DATA JPACKT /6/,              JPACKH /7/
      DATA JBLENT /516/,            JBLENH /516/
      DATA JBLINE /256/
C * STANDARD COMM DATA STATEMENTS
      DATA JPSTRG /63,32/
      DATA JPRMOD /0/
      DATA JEOFST /0/
      DATA JEOMC1,JEOMC2 /13,0/
      DATA JIGDEL /0/
      DATA JRESLU /12/
      DATA JXMTLM /2400/
      DATA JXMTDL /100/
      DATA JBREAK /200/
      DATA JFLAG  /3/
      DATA JQUEUE /1000/
      DATA JEOLST /13/
      DATA JBYPAS /10/
      DATA JLINLN /256/
      DATA JREOMF /1/
```

```
C * SET THE FIRST CHARACTER IN THE COMMAND ARRAY COMMON TO <ESC>
      JCODE(1)=27
C * SET I/O BUFFER POINTERS
      KOUTPT=0
      KINPT=0
      KINEND=0
C * INITIALIZE XY CHARACTERS
      DO 10 I=1,5
      KCHARS(I)=0
10    CONTINUE
C * SET OUTPUT BUFFER SIZE (SHOULD BE AS LARGE AS POSSIBLE)
C * (ALSO SET IN BLKEND)
      KBUFSZ=256
C * SET TERMINAL AND PLOTTER BLOCK MODES TO OFF
      KBARM=0
      KBMODE=0
      KBPLOT=0
      KBPSW=0
C * SET ALL STANDARD COMM PARAMETERS FIRST
      CALL PSTRNG (2,JPSTRG)
      CALL PROMPT (JPRMOD)
      CALL EOMCHR (JEOMC1,JEOMC2)
      CALL EOFSTG (0,JEOFST)
      CALL IGNDEL (JIGDEL)
      CALL XMTDLY (JXMTDL)
      CALL XMTLMT (JXMTLM)
      CALL BREAKT (JBREAK)
      CALL FLAG (JFLAG)
      CALL QUEUES (JQUEUE)
      CALL EOLSTG (1,JEOLST)
      CALL BYPCAN (JBYPAS)
      CALL LINLEN (JLINLN)
      CALL REOMF (JREOMF)
C * SET INTERNAL RESOLUTION FLAG TO 12-BIT (10-BIT IS FASTER BUT UGLIER)
      CALL RESLUT (JRESLU)
C * SET ALL BLOCK MODE PARAMETERS
      CALL BHEADR (5,JHEADT,5,JHEADH)
      CALL BNONCR (1,JNXMTT,6,JNXMTH)
      CALL BMASTC (JMASTT,JMASTH)
      CALL BCONTC (JCONTT,JCONTH)
      CALL BENDCH (JENDT,JENDH)
      CALL BLKLEN (JBLENT,JBLENH)
      CALL BLINE (JBLINE)
      CALL BPACK (JBYTET,JPACKT,JBYTEH,JPACKH)
      CALL BTOUT (5)
C * ARM BLOCK MODE FOR USE AFTER CALL TO BLOKGO
C       CALL BLKARM (1)
      RETURN
      END
```

```
C
C----------SUBROUTINE--IOEND---
C
      SUBROUTINE IOEND
C * IOEND SHUTS DOWN THE SOFTWARE I/O SYSTEM
      COMMON /COMM/ KOUTPT,KOUTBF(512),KINEND,KINPT,KINBUF(512),
     & KRESLU,KIGDEL,KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),
     & KEOMC1,KEOMC2,KCHARS(4),KBUFSZ,KBPLOT,KBPSW,
     & KBARM,KBMODE,KHLENH,KHEADH(10),KHLENT,KHEADT(10),KCONTH,
     & KCONTT,KENDH,KENDT,KNXNOH,KNXMTH(20),KNXNOT,KNXMTT(20),
     & KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,KPACKT,KBLENH,KBLENT,
     & KBLINE,KBLOKH,KEOPH,KEOMH,KEOFH,KEOMT,KEOFT
C * TURN OFF BLOCK MODE
      CALL BLKEND (2)
      CALL BLKARM (0)
C * TURN OFF PROMPT MODE IF IT WAS ON
      IF (KPRMOD.GT.0) CALL PROMPT (0)
C * CLEAR THE OUTPUT BUFFER
      CALL DUMP
      RETURN
      END
C
C----------SUBROUTINE--KYBDIN---
C
      SUBROUTINE KYBDIN (IREQ,IREC,ISTRNG)
C * KYBDIN GETS KEYBOARD INPUT, EXITTING AND REENTERING
C * TERMINAL AND PLOTTER BLOCK MODES IF NECESSARY
C * IREQ   - NUMBER OF CHARACTERS REQUESTED
C * IREC   - NUMBER OF CHARACTERS RECEIVED <= IREQ
C * ISTRNG - STRING FOR ADE CHARACTERS
C * KBMODE - S/W BLOCK MODE FLAG
C * JBMODE - TEMPORARY FOR REMEMBERING KBMODE
C * KBPLOT - PLOTTER BLOCK MODE FLAG
C * JBPLOT - TEMPORARY FOR REMEMBERING KBPLOT
      COMMON /COMM/ KOUTPT,KOUTBF(512),KINEND,KINPT,KINBUF(512),
     & KRESLU,KDELOY,KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),
     & KEOMC1,KEOMC2,KCHARS(4),KBUFSZ,KBPLOT,KBPSW,
     & KBARM,KBMODE,KHLENH,KHEADH(10),KHLENT,KHEADT(10),KCONTH,
     & KCONTT,KENDH,KENDT,KNXNOH,KNXMTH(20),KNXNOT,KNXMTT(20),
     & KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,KPACKT,KBLENH,KBLENT,
     & KBLINE,KBLOKH,KEOPH,KEOMH,KEOFH,KEOMT,KEOFT
      DIMENSION ISTRNG(1)
      JBMODE=KBMODE
      JBPLOT=KBPLOT
      KBPLOT=0
      CALL BLKEND (2)
      CALL STIN (IREQ,IREC,ISTRNG)
      IF (JBMODE.EQ.1) CALL BLOKGO
      KBPLOT=JBPLOT
      RETURN
      END
```

```
C
C----------SUBROUTINE--PLTRBM---
C
      SUBROUTINE PLTRBM (IBMODE)
C * PLTRBM SETS THE PLOTTER BLOCK MODE FLAG
C * IBMODE - MODE TO BE ESTABLISHED: 0=CONTINUOUS, 1=PLOTTER BLOCK
C * KBPLOT - NEW MODE TO BE USED AFTER NEXT DUMP
C * KBPSW  - MODE CURRENTLY IN USE BY SOFTWARE
      COMMON /COMM/ KOUTPT,KOUTBF(512),KINEND,KINPT,KINBUF(512),
     & KRESLU,KDELOY,KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),
     & KEOMC1,KEOMC2,KCHARS(4),KBUFSZ,KBPLOT,KBPSW,
     & KBARM,KBMODE,KHLENH,KHEADH(10),KHLENT,KHEADT(10),KCONTH,
     & KCONTT,KENDH,KENDT,KNXNOH,KNXMTH(20),KNXNOT,KNXMTT(20),
     & KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,KPACKT,KBLENH,KBLENT,
     & KBLINE,KBLOKH,KEOPH,KEOMH,KEOFH,KEOMT,KEOFT
      COMMON /COMRAY/ JCODE(256)
C * BRING VALUE WITHIN VALID RANGE
      JBPLOT=MIN0(1,MAX0(0,IBMODE))
C * EXIT IF NO CHANGE IN MODES
      IF (JBPLOT.EQ.KBPSW) RETURN
C * IF ENTERING PLOTTER BLOCK MODE, ENABLE PLOTTER
      IF (KBPLOT.NE.1) GO TO 10
      JCODE(2)=65
      JCODE(3)=69
      CALL STOUT (3,JCODE)
      KBPLOT=JBPLOT
      CALL DUMP
      KBUFSZ=KBUFSZ-11
      RETURN
C * IF EXITTING PLOTTER BLOCK MODE, DISABLE PLOTTER
10    KBPLOT=JBPLOT
      CALL DUMP
      KBUFSZ=KBUFSZ+11
      JCODE(2)=65
      JCODE(3)=70
      CALL STOUT (3,JCODE)
      RETURN
      END
```

```
C
C----------SUBROUTINE--FILIN---
C
      SUBROUTINE FILIN (IREQ,IREC,ISTRNG,IEOF)
C * FILIN INPUTS LINES FROM A TERMINAL FILE, SCANNING FOR EOF
C * IREQST - NUMBER OF CHARS REQUESTED
C * IRECVD - NUMBER OF CHARS RECEIVED
C * ISTRNG - CALLERS INPUT ARRAY
C * IEOF    - 1 IF EOF DETECTED, 0 IF NOT
      COMMON /COMM/ KOUTPT,KOUTBF(512),KINEND,KINPT,KINBUF(512),
     & KRESLU,KDELOY,KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),
     & KEOMC1,KEOMC2,KCHARS(4),KBUFSZ,KBPLOT,KBPSW,
     & KBARM,KBMODE,KHLENH,KHEADH(10),KHLENT,KHEADT(10),KCONTH,
     & KCONTT,KENDH,KENDT,KNXNOH,KNXMTH(20),KNXNOT,KNXMTT(20),
     & KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,KPACKT,KBLENH,KBLENT,
     & KBLINE,KBLOKH,KEOPH,KEOMH,KEOFH,KEOMT,KEOFT
      DIMENSION ISTRNG(1)
C * ZERO THE RETURN LENGTH
      IREC=0
C * BRANCH IF BUFFER HAS CHARACTERS IN IT
      IF (KINEND.GT.0) GO TO 40
C * BRANCH TO GET MORE INPUT VIA BLOCK MODE
      IF (KBMODE.EQ.1) GO TO 30
C * DUMP THE OUTPUT BUFFER TO BE SURE COMMANDS ARE SENT
      CALL DUMP
C * SIMULATE PROMPT FOR NON-PROMPTING SYSTEMS
C * REMOVE THIS LINE FOR PROMPTING SYSTEMS
      IF (KPRMOD.GT.0) CALL ADEOUT (KPRLEN,KPSTRG)
C * GET LINE OF INPUT FROM TERMINAL
      CALL ADEIN (KINEND,KINBUF)
      KINPT=0
C * BACK SCAN FOR EOF STRING
      KEOFT=0
      IF (KEOFLN.LE.0 .OR. KINEND.LT.KEOFLN) GO TO 40
      JINEND=KINEND
      JEOFPT=KEOFLN
10    IF (JEOFPT.EQ.0) GO TO 20
      IF (KINBUF(JINEND).NE.KEOFST(JEOFPT)) GO TO 40
      JINEND=JINEND-1
      JEOFPT=JEOFPT-1
      GO TO 10
20    KINEND=JINEND
      KEOFT=1
      GO TO 20
C * GET INPUT VIA BLOCK MODE
30    KEOMH=1
      CALL BLOKIO
      KEOMH=0
      KINPT=0
```

```
C * MOVE CHARACTERS FROM INPUT BUFFER TO USER ARRAY
C * EXIT IF REQUEST SATISFIED
40    IF (IREC.GE.IREQ) GO TO 70
C * CHECK FOR EMPTY BUFFER
      IF (KINPT.GE.KINEND) GO TO 50
      IREC=IREC+1
      KINPT=KINPT+1
      ISTRNG(IREC)=KINBUF(KINPT)
      GO TO 40
C * PAD WITH BLANKS
50    JFIRST=IREC+1
      DO 60 I=JFIRST,IREQ
      ISTRNG(I)=32
60    CONTINUE
C * EXIT IF BUFFER NOT EMPTY
70    IF (KINPT.LT.KINEND) GO TO 80
C * ZERO THE INPUT BUFFER
      KINEND=0
      KINPT=0
C * SET END-OF-FILE ONLY IF NO MORE CHARS IN BUFFER
80    IEOF=0
      IF (KEOFT.EQ.1 .AND. KINEND.EQ.0) IEOF=1
      RETURN
      END
C
C----------SUBROUTINE--SNDEOF---
C
      SUBROUTINE SNDEOF
C * SNDEOF SENDS AN END-OF-FILE INDICATOR TO THE TERMINAL
      COMMON /COMM/ KOUTPT,KOUTBF(512),KINEND,KINPT,KINBUF(512),
     & KRESLU,KDELOY,KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),
     & KEOMC1,KEOMC2,KCHARS(4),KBUFSZ,KBPLOT,KBPSW,
     & KBARM,KBMODE,KHLENH,KHEADH(10),KHLENT,KHEADT(10),KCONTH,
     & KCONTT,KENDH,KENDT,KNXNOH,KNXMTH(20),KNXNOT,KNXMTT(20),
     & KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,KPACKT,KBLENH,KBLENT,
     & KBLINE,KBLOKH,KEOPH,KEOMH,KEOFH,KEOMT,KEOFT
C * BRANCH FOR BLOCK MODE
      IF (KBMODE .EQ. 1) GO TO 20
C * SEND EOF STRING TO TERMINAL
      IF (KEOFLN.GT.0) CALL STOUT (KEOFLN,KEOFST)
      RETURN
C * FOR BLOCK MODE, SET EOF BIT AND DUMP BUFFER
20    KEOFH=1
      CALL DUMP
      KEOFH=0
      RETURN
      END
```

```
C
C----------SUBROUTINE--BLOKGO---
C
      SUBROUTINE BLOKGO
C * BLOKGO STARTS BLOCK MODE IN OPERATION
      COMMON /COMM/ KOUTPT,KOUTBF(512),KINEND,KINPT,KINBUF(512),
     & KRESLU,KIGDEL,KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),
     & KEOMC1,KEOMC2,KCHARS(4),KBUFSZ,KBPLOT,KBPSW,
     & KBARM,KBMODE,KHLENH,KHEADH(10),KHLENT,KHEADT(10),KCONTH,
     & KCONTT,KENDH,KENDT,KNXNOH,KNXMTH(20),KNXNOT,KNXMTT(20),
     & KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,KPACKT,KBLENH,KBLENT,
     & KBLINE,KBLOKH,KEOPH,KEOMH,KEOFH,KEOMT,KEOFT
C * IF BLOCK MODE NOT ARMED, EXIT
      IF (KBARM.NE.1) RETURN
C * IF ALREADY IN BLOCK MODE, EXIT
      IF (KBMODE.EQ.1) RETURN
C * DUMP BUFFER TO GET CLEAN ENTRY INTO BLOCK MODE
      CALL DUMP
C * SET CONTROL BYTE BITS
      KBLOKH=1
      KEOPH=0
      KEOFH=0
      KEOMH=0
C * PUT SOFTWARE INTO BLOCK PROTOCOL
      KBMODE=1
C * CHANGE OUTPUT BUFFER SIZE TO BLOCK SIZE
      KBUFSZ=KBLENH-4
      IF (KBPSW.EQ.1) KBUFSZ=KBUFSZ-11
C * TURN TERMINAL ECHO ON
      CALL ECHO (1)
      RETURN
      END
```

```
C
C----------SUBROUTINE--BLKEND---
C
      SUBROUTINE BLKEND (IACK)
C * BLKEND ENDS THE BLOCK MODE PROTOCOL BUT DOES NOT DISARM BLOCK MODE
      COMMON /COMM/ KOUTPT,KOUTBF(512),KINEND,KINPT,KINBUF(512),
     & KRESLU,KIGDEL,KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),
     & KEOMC1,KEOMC2,KCHARS(4),KBUFSZ,KBPLOT,KBPSW,
     & KBARM,KBMODE,KHLENH,KHEADH(10),KHLENT,KHEADT(10),KCONTH,
     & KCONTT,KENDH,KENDT,KNXNOH,KNXMTH(20),KNXNOT,KNXMTT(20),
     & KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,KPACKT,KBLENH,KBLENT,
     & KBLINE,KBLOKH,KEOPH,KEOMH,KEOFH,KEOMT,KEOFT
C * IF NOT IN BLOCK MODE, EXIT
      IF (KBMODE.EQ.0) RETURN
C * TURN OFF TERMINAL ECHO (IF IT SHOULD NOT BE ON NORMALLY)
      CALL ECHO (0)
C * SET THE CONTROL BITS TO END THE BLOCK PROTOCOL
      KBLOKH=0
      KEOPH=MINO(3,MAXO(2,IACK))
C * DUMP THE BUFFER AND TURN OFF SOFTWARE BLOCK MODE
      CALL DUMP
      KBMODE=0
C * RESTORE OUTPUT BUFFER SIZE
      KBUFSZ=256
      IF (KBPSW.EQ.1) KBUFSZ=KBUFSZ-11
      RETURN
      END
C
C-----------SUBROUTINE--BLKARM---
C
      SUBROUTINE BLKARM (IARM)
C * BLKARM ARMS/DISARMS BLOCK MODE
C * IARM   - BLOCK MODE ARM/DISARM
C * JCODE  - COMMAND ARRAY
C * LEN    - LENGTH OF IONOFF AS PACKED INTEGER
      COMMON /COMM/ KOUTPT,KOUTBF(512),KINEND,KINPT,KINBUF(512),
     & KRESLU,KIGDEL,KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),
     & KEOMC1,KEOMC2,KCHARS(4),KBUFSZ,KBPLOT,KBPSW,
     & KBARM,KBMODE,KHLENH,KHEADH(10),KHLENT,KHEADT(10),KCONTH,
     & KCONTT,KENDH,KENDT,KNXNOH,KNXMTH(20),KNXNOT,KNXMTT(20),
     & KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,KPACKT,KBLENH,KBLENT,
     & KBLINE,KBLOKH,KEOPH,KEOMH,KEOFH,KEOMT,KEOFT
      COMMON /COMRAY/ JCODE(256)
      DIMENSION JINRAY(5),JENQ(2)
      DATA JENQ/27,5/
      JCODE(2)=79
      JCODE(3)=66
      CALL INTPAK (IARM,LEN,JCODE(4))
      CALL STOUT (LEN+3,JCODE)
      KBARM=MINO(1,MAXO(0,IARM))
C * REQUEST STATUS TO ALLOW TERMINAL TIME TO CHANGE MODES
      CALL STOUT (2,JENQ)
      CALL EMPTIN
      CALL STIN (5,JREC,JINRAY)
      RETURN
      END
```

```
C
C----------SUBROUTINE--BLKLEN---
C
      SUBROUTINE BLKLEN (IBLENT,IBLENH)
C * BLCOKL SETS THE BLOCK LENGTHS
C * IBLENT - TERMINAL'S BLOCK LENGTH
C * IBLENH - HOST'S BLOCK LENGTH
C * JCODE  - COMMAND ARRAY
C * LEN1   - LENGTH OF IBLENT AS PACKED INTEGER
C * LEN2   - LENGTH OF IBLENH AS PACKED INTEGER
       COMMON /COMM/ KOUTPT,KOUTBF(512),KINEND,KINPT,KINBUF(512),
      & KRESLU,KIGDEL,KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),
      & KEOMC1,KEOMC2,KCHARS(4),KBUFSZ,KBPLOT,KBPSW,
      & KBARM,KBMODE,KHLENH,KHEADH(10),KHLENT,KHEADT(10),KCONTH,
      & KCONTT,KENDH,KENDT,KNXNOH,KNXMTH(20),KNXNOT,KNXMTT(20),
      & KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,KPACKT,KBLENH,KBLENT,
      & KBLINE,KBLOKH,KEOPH,KEOMH,KEOFH,KEOMT,KEOFT
       COMMON /COMRAY/ JCODE(256)
       JCODE(2)=79
       JCODE(3)=83
       KBLENH=IBLENH
       KBLENT=IBLENT
       CALL INTPAK (IBLENT,LEN1,JCODE(4))
       CALL INTPAK (IBLENH,LEN2,JCODE(LEN1+4))
       CALL STOUT (LEN1+LEN2+3,JCODE)
       RETURN
       END
C
C----------SUBROUTINE--BENDCH---
C
      SUBROUTINE BENDCH (IENDT,IENDH)
C * BENDCH SETS THE BLOCK END CHARACTERS
C * IENDT - TERMINAL'S BLOCK END CHARACTER
C * IENDH - HOST'S BLOCK END CHARACTER
C * JCODE  - COMMAND ARRAY
C * LEN1   - LENGTH OF IENDT AS PACKED INTEGER
C * LEN2   - LENGTH OF IENDH AS PACKED INTEGER
       COMMON /COMM/ KOUTPT,KOUTBF(512),KINEND,KINPT,KINBUF(512),
      & KRESLU,KIGDEL,KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),
      & KEOMC1,KEOMC2,KCHARS(4),KBUFSZ,KBPLOT,KBPSW,
      & KBARM,KBMODE,KHLENH,KHEADH(10),KHLENT,KHEADT(10),KCONTH,
      & KCONTT,KENDH,KENDT,KNXNOH,KNXMTH(20),KNXNOT,KNXMTT(20),
      & KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,KPACKT,KBLENH,KBLENT,
      & KBLINE,KBLOKH,KEOPH,KEOMH,KEOFH,KEOMT,KEOFT
       COMMON /COMRAY/ JCODE(256)
       JCODE(2)=79
       JCODE(3)=69
       KENDH=IENDH
       KENDT=IENDT
       CALL INTPAK (IENDT,LEN1,JCODE(4))
       CALL INTPAK (IENDH,LEN2,JCODE(LEN1+4))
       CALL STOUT (LEN1+LEN2+3,JCODE)
       RETURN
       END
```

```
C
C----------SUBROUTINE--BCONTC---
C
      SUBROUTINE BCONTC (ICONTT,ICONTH)
C * BCONTC SETS THE BLOCK CONTINUE CHARACTERS
C * ICONTT - TERMINAL'S BLOCK CONTINUE CHARACTER
C * ICONTH - HOST'S BLOCK CONTINUE CHARACTER
C * JCODE  - COMMAND ARRAY
C * LEN1   - LENGTH OF ICONTT AS PACKED INTEGER
C * LEN2   - LENGTH OF ICONTH AS PACKED INTEGER
      COMMON /COMM/ KOUTPT,KOUTBF(512),KINEND,KINPT,KINBUF(512),
     & KRESLU,KIGDEL,KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),
     & KEOMC1,KEOMC2,KCHARS(4),KBUFSZ,KBPLOT,KBPSW,
     & KBARM,KBMODE,KHLENH,KHEADH(10),KHLENT,KHEADT(10),KCONTH,
     & KCONTT,KENDH,KENDT,KNXNOH,KNXMTH(20),KNXNOT,KNXMTT(20),
     & KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,KPACKT,KBLENH,KBLENT,
     & KBLINE,KBLOKH,KEOPH,KEOMH,KEOFH,KEOMT,KEOFT
      COMMON /COMRAY/ JCODE(256)
      JCODE(2)=79
      JCODE(3)=67
      KCONTH=ICONTH
      KCONTT=ICONTT
      CALL INTPAK (ICONTT,LEN1,JCODE(4))
      CALL INTPAK (ICONTH,LEN2,JCODE(LEN1+4))
      CALL STOUT (LEN1+LEN2+3,JCODE)
      RETURN
      END
C
C----------SUBROUTINE--BMASTC---
C
      SUBROUTINE BMASTC (IMASTT,IMASTH)
C * BMASTC SETS THE BLOCK MASTER CHARACTERS
C * IMASTT - TERMINAL'S MASTER CHARACTER
C * IMASTH - HOST'S MASTER CHARACTER
C * JCODE  - COMMAND ARRAY
C * LEN1   - LENGTH OF IMASTT AS PACKED INTEGER
C * LEN2   - LENGTH OF IMASTH AS PACKED INTEGER
      COMMON /COMM/ KOUTPT,KOUTBF(512),KINEND,KINPT,KINBUF(512),
     & KRESLU,KIGDEL,KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),
     & KEOMC1,KEOMC2,KCHARS(4),KBUFSZ,KBPLOT,KBPSW,
     & KBARM,KBMODE,KHLENH,KHEADH(10),KHLENT,KHEADT(10),KCONTH,
     & KCONTT,KENDH,KENDT,KNXNOH,KNXMTH(20),KNXNOT,KNXMTT(20),
     & KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,KPACKT,KBLENH,KBLENT,
     & KBLINE,KBLOKH,KEOPH,KEOMH,KEOFH,KEOMT,KEOFT
      COMMON /COMRAY/ JCODE(256)
      JCODE(2)=79
      JCODE(3)=77
      KMASTH=IMASTH
      KMASTT=IMASTT
      CALL INTPAK (IMASTT,LEN1,JCODE(4))
      CALL INTPAK (IMASTH,LEN2,JCODE(LEN1+4))
      CALL STOUT (LEN1+LEN2+3,JCODE)
      RETURN
      END
```

```
C
C-----------SUBROUTINE--BPACK---
C
      SUBROUTINE BPACK (IBST,IPKT,IBSH,IPKH)
C * BPACK SETS THE BIT PACKING VALUES
C * IBST   - TERMINAL'S UNPACKED BITS/CHAR
C * IPKT   - TERMINAL'S PACKED BITS/CHAR
C * IBSH   - HOST'S UNPACKED BITS/CHAR
C * IPKH   - HOST'S PACKED BITS/CHAR
C * JCODE  - COMMAND ARRAY
C * LEN1   - LENGTH OF PARAMETER AS PACKED INTEGER
C * LEN2   - TOTAL LENGTH OF COMMAND
      COMMON /COMM/ KOUTPT,KOUTBF(512),KINEND,KINPT,KINBUF(512),
     & KRESLU,KIGDEL,KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),
     & KEOMC1,KEOMC2,KCHARS(4),KBUFSZ,KBPLOT,KBPSW,
     & KBARM,KBMODE,KHLENH,KHEADH(10),KHLENT,KHEADT(10),KCONTH,
     & KCONTT,KENDH,KENDT,KNXNOH,KNXMTH(20),KNXNOT,KNXMTT(20),
     & KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,KPACKT,KBLENH,KBLENT,
     & KBLINE,KBLOKH,KEOPH,KEOMH,KEOFH,KEOMT,KEOFT
      COMMON /COMRAY/ JCODE(256)
      JCODE(2)=79
      JCODE(3)=80
      KBYTEH=2**IBSH
      KBYTET=2**IBST
      KPACKH=2**IPKH
      KPACKT=2**IPKT
      LEN2=4
      CALL INTPAK (IBST,LEN1,JCODE(LEN2))
      LEN2=LEN2+LEN1
      CALL INTPAK (IPKT,LEN1,JCODE(LEN2))
      LEN2=LEN2+LEN1
      CALL INTPAK (IBSH,LEN1,JCODE(LEN2))
      LEN2=LEN2+LEN1
      CALL INTPAK (IPKH,LEN1,JCODE(LEN2))
      LEN2=LEN2+LEN1-1
      CALL STOUT (LEN2,JCODE)
      RETURN
      END
C
C-----------SUBROUTINE--BTOUT---
C
      SUBROUTINE BTOUT (ITIME)
C * BTOUT SETS THE TERMINAL'S BLOCK TIMEOUT
C * ITIME  - SECONDS BEFORE RETRANSMITTING BLOCK
C * JCODE  - COMMAND ARRAY
C * LEN    - LENGTH OF ITIME AS PACKED INTEGER
      COMMON /COMRAY/ JCODE(256)
      JCODE(2)=79
      JCODE(3)=84
      CALL INTPAK (ITIME,LEN,JCODE(4))
      CALL STOUT (LEN+3,JCODE)
      RETURN
      END
```

```
C
C------------SUBROUTINE--BNONCR---
C
      SUBROUTINE BNONCR (INXNOT,INXMTT,INXNOH,INXMTH)
C * BNONCR SETS THE BLOCK NON-TRANSMITTABLE CHARACTERS
C * INXNOT - NUMBER OF TERMINAL NON-XMT CHARS
C * INXMTT - ARRAY OF TERMINAL'S NON-XMT CHARS
C * INXNOH - NUMBER OF HOST'S NON-XMT CHARS
C * INXMTH - ARRAY OF HOST'S NON-XMT CHARS
C * JCODE  - COMMAND ARRAY
C * LEN1   - LENGTH OF TERMINAL ARRAY AS PACKED INTEGERS
C * LEN2   - LENGTH OF HOST ARRAY AS PACKED INTEGERS
      COMMON /COMM/ KOUTPT,KOUTBF(512),KINEND,KINPT,KINBUF(512),
     & KRESLU,KIGDEL,KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),
     & KEOMC1,KEOMC2,KCHARS(4),KBUFSZ,KBPLOT,KBPSW,
     & KBARM,KBMODE,KHLENH,KHEADH(10),KHLENT,KHEADT(10),KCONTH,
     & KCONTT,KENDH,KENDT,KNXNOH,KNXMTH(20),KNXNOT,KNXMTT(20),
     & KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,KPACKT,KBLENH,KBLENT,
     & KBLINE,KBLOKH,KEOPH,KEOMH,KEOFH,KEOMT,KEOFT
      COMMON /COMRAY/ JCODE(256)
      DIMENSION INXMTT(1),INXMTH(1)
      JCODE(2)=79
      JCODE(3)=78
      KNXNOH=MIN0(20,MAX0(0,INXNOH))
      IF (KNXNOH.EQ.0) GO TO 20
      DO 10 I=1,KNXNOH
10    KNXMTH(I)=INXMTH(I)
20    KNXNOT=MIN0(20,MAX0(0,INXNOT))
      IF (KNXNOT.EQ.0) GO TO 40
      DO 30 I=1,KNXNOT
30    KNXMTT(I)=INXMTT(I)
40    CALL INTRAY (INXNOT,INXMTT,LEN1,JCODE(4))
      CALL INTRAY (INXNOH,INXMTH,LEN2,JCODE(LEN1+4))
      CALL STOUT (LEN1+LEN2+3,JCODE)
      RETURN
      END
```

```
C
C------------SUBROUTINE--BHEADR---
C
      SUBROUTINE BHEADR (IHLENT,IHEADT,IHLENH,IHEADH)
C * BHEADR SETS THE BLOCK HEADERS
C * IHLENT - LENGTH OF TERMINAL'S HEADER
C * IHEADT - TERMINAL'S HEADER
C * IHLENH - LENGTH OF HOST'S HEADER
C * IHEADH - HOST'S HEADER
C * JCODE  - COMMAND ARRAY
C * LEN1   - LENGTH OF TERMINAL HEADER AS PACKED INTEGERS
C * LEN2   - LENGTH OF HOST HEADER AS PACKED INTEGERS
       COMMON /COMM/ KOUTPT,KOUTBF(512),KINEND,KINPT,KINBUF(512),
     & KRESLU,KIGDEL,KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),
     & KEOMC1,KEOMC2,KCHARS(4),KBUFSZ,KBPLOT,KBPSW,
     & KBARM,KBMODE,KHLENH,KHEADH(10),KHLENT,KHEADT(10),KCONTH,
     & KCONTT,KENDH,KENDT,KNXNOH,KNXMTH(20),KNXNOT,KNXMTT(20),
     & KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,KPACKT,KBLENH,KBLENT,
     & KBLINE,KBLOKH,KEOPH,KEOMH,KEOFH,KEOMT,KEOFT
       COMMON /COMRAY/ JCODE(256)
       DIMENSION IHEADT(1),IHEADH(1)
       JCODE(2)=79
       JCODE(3)=72
       KHLENH=MIN0(20,MAX0(0,IHLENH))
       IF (KHLENH.EQ.0) GO TO 20
       DO 10 I=1,KHLENH
10     KHEADH(I)=IHEADH(I)
20     KHLENT=MIN0(20,MAX0(0,IHLENT))
       IF (KHLENT.EQ.0) GO TO 40
       DO 30 I=1,KHLENT
30     KHEADT(I)=IHEADT(I)
40     CALL INTRAY (IHLENT,IHEADT,LEN1,JCODE(4))
       CALL INTRAY (IHLENH,IHEADH,LEN2,JCODE(LEN1+4))
       CALL STOUT (LEN1+LEN2+3,JCODE)
       RETURN
       END
```

```
C
C----------SUBROUTINE--BLINE---
C
      SUBROUTINE BLINE (ILENTH)
C * BLINE SETS THE BLOCK LINE LENGTH
C * ILENTH - BLOCK LINE LENGTH
C * JCODE  - COMMAND ARRAY
C * LEN    - LENGTH OF ILENTH AS PACKED INTEGER
      COMMON /COMM/ KOUTPT,KOUTBF(512),KINEND,KINPT,KINBUF(512),
     & KRESLU,KIGDEL,KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),
     & KEOMC1,KEOMC2,KCHARS(4),KBUFSZ,KBPLOT,KBPSW,
     & KBARM,KBMODE,KHLENH,KHEADH(10),KHLENT,KHEADT(10),KCONTH,
     & KCONTT,KENDH,KENDT,KNXNOH,KNXMTH(20),KNXNOT,KNXMTT(20),
     & KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,KPACKT,KBLENH,KBLENT,
     & KBLINE,KBLOKH,KEOPH,KEOMH,KEOFH,KEOMT,KEOFT
      COMMON /COMRAY/ JCODE(256)
      JCODE(2)=79
      JCODE(3)=76
      CALL INTPAK (ILENTH,LEN,JCODE(4))
      CALL STOUT (LEN+3,JCODE)
      KBLINE=MIN0(256,MAX0(50,ILENTH))
      RETURN
      END
C
C----------SUBROUTINE--ECHO---
C
      SUBROUTINE ECHO (IECHO)
C * ECHO TURNS THE TERMINAL ECHO ON OR OFF
C * IECHO  - 0 FOR OFF, 1 FOR ON
      COMMON /COMRAY/ JCODE(256)
      JCODE(2)=75
      JCODE(3)=69
      CALL INTPAK (IECHO,LEN,JCODE(4))
      CALL STOUT (LEN+3,JCODE)
      RETURN
      END
C
C-----------SUBROUTINE--FLAG---
C
      SUBROUTINE FLAG (IFLAG)
C * FLAG SETS THE FLAGGING MODE AT THE TERMINAL
C * IMODE  - FLAGGING MODE [0,5]
C * JCODE  - COMMAND ARRAY
C * LEN    - IMODE AS PACKED INTEGER
      COMMON /COMRAY/ JCODE(256)
      JCODE(2)=78
      JCODE(3)=70
      CALL INTPAK (IFLAG,LEN,JCODE(4))
      CALL STOUT (LEN+3,JCODE)
      RETURN
      END
```

```
C
C-----------SUBROUTINE--XMTLMT---
C
      SUBROUTINE XMTLMT (LIMIT)
C * XMTLMT SETS THE TERMINAL-TO-HOST TRANSMIT LIMIT
C * LIMIT  - BAUD RATE LIMIT [110,19200]
C * JCODE  - COMMAND ARRAY
C * LEN    - LENGTH OF LIMIT AS PACKED INTEGER
      COMMON /COMRAY/ JCODE(256)
      JCODE(2)=78
      JCODE(3)=76
      CALL INTPAK (LIMIT,LEN,JCODE(4))
      CALL STOUT (LEN+3,JCODE)
      RETURN
      END
C
C-----------SUBROUTINE--BREAKT---
C
      SUBROUTINE BREAKT (IDELAY)
C * BREAKT SETS THE TERMINALS BREAK TIME DELAY
C * IDELAY - BREAK DELAY IN MILLISECONDS
C * JCODE  - COMMAND ARRAY
C * LEN    - LEN OF IDELAY AS PACKED INTEGER
      COMMON /COMRAY/ JCODE(256)
      JCODE(2)=78
      JCODE(3)=75
      CALL INTPAK (IDELAY,LEN,JCODE(4))
      CALL STOUT (LEN+3,JCODE)
      RETURN
      END
C
C-----------SUBROUTINE--PROMPT---
C
      SUBROUTINE PROMPT (IONOFF)
C * PROMPT TURNS TERMINAL PROMPT MODE ON AND OFF
C * IONOFF - PROMT MODE ON/OFF [0,2]
C * JCODE  - COMMAND ARRAY
C * LEN    - LENGTH OF IONOFF AS PACKED INTEGER
      COMMON /COMM/ KOUTPT,KOUTBF(512),KINEND,KINPT,KINBUF(512),
     & KRESLU,KDELOY,KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),
     & KEOMC1,KEOMC2,KCHARS(4),KBUFSZ,KBPLOT,KBPSW,
     & KBARM,KBMODE,KHLENH,KHEADH(10),KHLENT,KHEADT(10),KCONTH,
     & KCONTT,KENDH,KENDT,KNXNOH,KNXMTH(20),KNXNOT,KNXMTT(20),
     & KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,KPACKT,KBLENH,KBLENT,
     & KBLINE,KBLOKH,KEOPH,KEOMH,KEOFH,KEOMT,KEOFT
      COMMON /COMRAY/ JCODE(256)
      JCODE(2)=78
      JCODE(3)=77
      CALL INTPAK(IONOFF,LEN,JCODE(4))
      CALL STOUT(LEN+3,JCODE)
      KPRMOD=IONOFF
      RETURN
      END
```

```
C
C-----------SUBROUTINE--EOMCHR---
C
      SUBROUTINE EOMCHR (ICHAR1,ICHAR2)
C * EOMCHR SETS THE TERMINAL EOM CHARACTERS
C * ICHAR1 - FIRST EOM CHARACTER
C * ICHAR2 - SECOND EOM CHARACTER
C * JCODE  - COMMAND ARRAY
C * LEN1   - LENGTH OF ICHAR1 AS PACKED INTEGER
C * LEN2   - LENGTH OF ICHAR2 AS PACKED INTEGER
      COMMON /COMM/ KOUTPT,KOUTBF(512),KINEND,KINPT,KINBUF(512),
     & KRESLU,KDELOY,KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),
     & KEOMC1,KEOMC2,KCHARS(4),KBUFSZ,KBPLOT,KBPSW,
     & KBARM,KBMODE,KHLENH,KHEADH(10),KHLENT,KHEADT(10),KCONTH,
     & KCONTT,KENDH,KENDT,KNXNOH,KNXMTH(20),KNXNOT,KNXMTT(20),
     & KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,KPACKT,KBLENH,KBLENT,
     & KBLINE,KBLOKH,KEOPH,KEOMH,KEOFH,KEOMT,KEOFT
      COMMON /COMRAY/ JCODE(256)
      JCODE(2)=78
      JCODE(3)=67
      CALL INTPAK (ICHAR1,LEN1,JCODE(4))
      CALL INTPAK (ICHAR2,LEN2,JCODE(LEN1+4))
      CALL STOUT (LEN1+LEN2+3,JCODE)
      KEOMC1=ICHAR1
      KEOMC2=ICHAR2
      RETURN
      END
C
C-----------SUBROUTINE--XMTDLY---
C
      SUBROUTINE XMTDLY (IMSEC)
C * XMTDLY SETS THE TERMINAL TRANSMIT DELAY IN MILLISECONDS
C * IMSEC  - MILLISECONDS OF DELAY
C * JCODE  - COMMAND ARRAY
C * LEN    - LENGTH OF IMSEC AS PACKED INTEGER
      COMMON /COMRAY/ JCODE(256)
      JCODE(2)=78
      JCODE(3)=68
      CALL INTPAK (IMSEC,LEN,JCODE(4))
      CALL STOUT (LEN+3,JCODE)
      RETURN
      END
```

```
C
C-----------SUBROUTINE--PSTRNG---
C
      SUBROUTINE PSTRNG (NUM,IARRAY)
C * PSTRNG SETS THE TERMINAL'S PROMPT STRING
C * NUM    - NUMBER OF CHARACTERS IN THE PROMPT STRING
C * IARRAY - ARRAY CONTAINING ADE OF PROMPT STRING
C * JCODE  - COMMAND ARRAY
C * LEN    - LENGTH OF PROMPT STRING AS PACKED INTEGERS
      COMMON /COMM/ KOUTPT,KOUTBF(512),KINEND,KINPT,KINBUF(512),
     & KRESLU,KDELOY,KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),
     & KEOMC1,KEOMC2,KCHARS(4),KBUFSZ,KBPLOT,KBPSW,
     & KBARM,KBMODE,KHLENH,KHEADH(10),KHLENT,KHEADT(10),KCONTH,
     & KCONTT,KENDH,KENDT,KNXNOH,KNXMTH(20),KNXNOT,KNXMTT(20),
     & KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,KPACKT,KBLENH,KBLENT,
     & KBLINE,KBLOKH,KEOPH,KEOMH,KEOFH,KEOMT,KEOFT
      COMMON /COMRAY/ JCODE(256)
      DIMENSION IARRAY(1)
      JCODE(2)=78
      JCODE(3)=83
      CALL INTRAY (NUM,IARRAY,LEN,JCODE(4))
      CALL STOUT (LEN+3,JCODE)
      KPRLEN=MINO(10,MAXO(NUM,0))
      IF (KPRLEN.EQ.0) RETURN
      DO 100 I=1,KPRLEN
      KPSTRG(I)=IARRAY(I)
100   CONTINUE
      RETURN
      END
C
C-----------SUBROUTINE--EOLSTG---
C
      SUBROUTINE EOLSTG (NUM,ICHAR)
C * EOLSTG SETS THE TERMINAL END-OF-LINE STRING
C * NUM    - NUMBER OF CHARACTERS IN THE EOL SEQUENCE (TERM-STRING)
C * ICHAR  - ARRAY HOLDING ADE OF EOL SEQUENCE (...)
C * JCODE  - COMMAND ARRAY
C * LEN    - LENGTH OF EOL SEQUENCE AS PACKED INTEGERS
      COMMON /COMRAY/ JCODE(256)
      JCODE(2)=78
      JCODE(3)=84
      CALL INTRAY (NUM,ICHAR,LEN,JCODE(4))
      CALL STOUT (LEN+3,JCODE)
      RETURN
      END
C
C-----------SUBROUTINE--BYPCAN---
C
      SUBROUTINE BYPCAN (ICHAR)
C * BYPCAN SETS THE TERMINAL'S BYPASS CANCEL CHARACTER
C * ICHAR  - BYPASS CANCEL CHARACTER
C * JCODE  - COMMAND ARRAY
C * LEN    - LENGTH OF ICHAR AS PACKED INTEGER
      COMMON /COMRAY/ JCODE(256)
      JCODE(2)=78
      JCODE(3)=85
      CALL INTPAK (ICHAR,LEN,JCODE(4))
      CALL STOUT (LEN+3,JCODE)
      RETURN
      END
```

```
C
C-----------SUBROUTINE--QUEUES---
C
      SUBROUTINE QUEUES (IBYTE)
C * QUEUES SETS THE TERMINAL'S INPUT QUEUESIZE
C * IBYTE  - SIZE OF QUEUE IN BYTES
C * JCODE  - COMMAND ARRAY
C * LEN    - LENGTH OF IBYTE AS PACKED INTEGER
      COMMON /COMRAY/ JCODE(256)
      JCODE(2)=78
      JCODE(3)=81
      CALL INTPAK (IBYTE,LEN,JCODE(4))
      CALL STOUT (LEN+3,JCODE)
      RETURN
      END
C
C-----------SUBROUTINE--EOFSTG---
C
      SUBROUTINE EOFSTG (NUM,IARRAY)
      COMMON /COMM/ KOUTPT,KOUTBF(512),KINEND,KINPT,KINBUF(512),
     & KRESLU,KDELOY,KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),
     & KEOMC1,KEOMC2,KCHARS(4),KBUFSZ,KBPLOT,KBPSW,
     & KBARM,KBMODE,KHLENH,KHEADH(10),KHLENT,KHEADT(10),KCONTH,
     & KCONTT,KENDH,KENDT,KNXNOH,KNXMTH(20),KNXNOT,KNXMTT(20),
     & KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,KPACKT,KBLENH,KBLENT,
     & KBLINE,KBLOKH,KEOPH,KEOMH,KEOFH,KEOMT,KEOFT
      COMMON /COMRAY/ JCODE(256)
      DIMENSION IARRAY(1)
      JCODE(2)=78
      JCODE(3)=69
      CALL INTRAY (NUM,IARRAY,LEN,JCODE(4))
      CALL STOUT (LEN+3,JCODE)
      KEOFLN=MIN0(10,MAX0(NUM,0))
      IF (KEOFLN.EQ.0) RETURN
      DO 100 I=1,KEOFLN
      KEOFST(I)=IARRAY(I)
100   CONTINUE
      RETURN
      END
C
C----------SUBROUTINE--IGNDEL---
C
      SUBROUTINE IGNDEL (IGNORE)
      COMMON /COMM/ KOUTPT,KOUTBF(512),KINEND,KINPT,KINBUF(512),
     & KRESLU,KDELOY,KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),
     & KEOMC1,KEOMC2,KCHARS(4),KBUFSZ,KBPLOT,KBPSW,
     & KBARM,KBMODE,KHLENH,KHEADH(10),KHLENT,KHEADT(10),KCONTH,
     & KCONTT,KENDH,KENDT,KNXNOH,KNXMTH(20),KNXNOT,KNXMTT(20),
     & KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,KPACKT,KBLENH,KBLENT,
     & KBLINE,KBLOKH,KEOPH,KEOMH,KEOFH,KEOMT,KEOFT
      COMMON /COMRAY/ JCODE(256)
      JCODE(2)=75
      JCODE(3)=73
      CALL INTPAK (IGNORE,LEN,JCODE(4))
      CALL STOUT (LEN+3,JCODE)
      KDELOY=MIN0(1,MAX0(0,IGNORE))
      RETURN
      END
```

```
C
C----------SUBROUTINE--LINLEN---
C
      SUBROUTINE LINLEN (ILEN)
C * LINLEN SENDS THE SET-MAX-LINE-LENGTH COMMAND
      COMMON /COMRAY/ JCODE(256)
      JCODE(2)=73
      JCODE(3)=76
      CALL INTPAK (ILEN,JLEN,JCODE(4))
      CALL STOUT (JLEN+3,JCODE)
      RETURN
      END
C
C----------SUBROUTINE--REOMF---
C
      SUBROUTINE REOMF (IREOMF)
C * REOMF SENDS SET-REPORT-EOM-FREQUENCY COMMAND
      COMMON /COMRAY/ JCODE(256)
      JCODE(2)=73
      JCODE(3)=77
      CALL INTPAK (IREOMF,JLEN,JCODE(4))
      CALL STOUT (JLEN+3,JCODE)
      RETURN
      END
C
C----------SUBROUTINE--RESLUT--
C
      SUBROUTINE RESLUT (IRESLU)
      COMMON /COMM/ KOUTPT,KOUTBF(512),KINEND,KINPT,KINBUF(512),
     & KRESLU,KDELOY,KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),
     & KEOMC1,KEOMC2,KCHARS(4),KBUFSZ,KBPLOT,KBPSW,
     & KBARM,KBMODE,KHLENH,KHEADH(10),KHLENT,KHEADT(10),KCONTH,
     & KCONTT,KENDH,KENDT,KNXNOH,KNXMTH(20),KNXNOT,KNXMTT(20),
     & KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,KPACKT,KBLENH,KBLENT,
     & KBLINE,KBLOKH,KEOPH,KEOMH,KEOFH,KEOMT,KEOFT
      KRESLU=12
      IF (IRESLU.LT.12) KRESLU=10
      RETURN
      END
C
C----------SUBROUTINE--EMPTIN--
C
      SUBROUTINE EMPTIN
C * EMPTIN ZEROES THE INPUT BUFFER
      COMMON /COMM/ KOUTPT,KOUTBF(512),KINEND,KINPT,KINBUF(512),
     & KRESLU,KIGDEL,KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),
     & KEOMC1,KEOMC2,KCHARS(4),KBUFSZ,KBPLOT,KBPSW,
     & KBARM,KBMODE,KHLENH,KHEADH(10),KHLENT,KHEADT(10),KCONTH,
     & KCONTT,KENDH,KENDT,KNXNOH,KNXMTH(20),KNXNOT,KNXMTT(20),
     & KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,KPACKT,KBLENH,KBLENT,
     & KBLINE,KBLOKH,KEOPH,KEOMH,KEOFH,KEOMT,KEOFT
      KINEND=0
      KINPT=0
      RETURN
      END
```

```
C
C----------SUBROUTINE--STIN-----
C
      SUBROUTINE STIN (IREQ,IREC,ISTRNG)
C * STIN RETURNS CHARACTERS FROM ITS INPUT BUFFER, GETTING MORE
C * INPUT FROM THE TERMINAL IF THE BUFFER IS EMPTY
C * IT PADS THE USER ARRAY WITH SPACES IF LESS THAN REQUESTED IS RECEIVED
C * IREQ   - NUMBER OF CHARACTERS REQUESTED
C * IREC   - NUMBER OF CHARACTERS ACTUALLY RETURNED
C * ISTRNG - ARRAY FOR RETURNED ADE STRING
C * KINEND - POINTER TO END OF INPUT IN KINBUF
C * KINPT  - POINTER TO CURRENT POSITION IN KINBUF
C * KINBUF - INPUT BUFFER
C * KPRMOD - PROMPT MODE FLAG: 0 - OFF, 1 - ON
C * KPRLEN - LENGTH OF PROMPT STRING
C * KPSTRG - ARRAY FOR PROMPT STRING
        COMMON /COMM/ KOUTPT,KOUTBF(512),KINEND,KINPT,KINBUF(512),
       & KRESLU,KIGDEL,KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),
       & KEOMC1,KEOMC2,KCHARS(4),KBUFSZ,KBPLOT,KBPSW,
       & KBARM,KBMODE,KHLENH,KHEADH(10),KHLENT,KHEADT(10),KCONTH,
       & KCONTT,KENDH,KENDT,KNXNOH,KNXMTH(20),KNXNOT,KNXMTT(20),
       & KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,KPACKT,KBLENH,KBLENT,
       & KBLINE,KBLOKH,KEOPH,KEOMH,KEOFH,KEOMT,KEOFT
        DIMENSION ISTRNG(1)
C * ZERO THE RETURN LENGTH
        IREC=0
C * BRANCH IF BUFFER HAS CHARACTERS IN IT
        IF (KINEND.GT.0) GO TO 20
C * BRANCH TO GET MORE INPUT VIA BLOCK MODE
        IF (KBMODE.EQ.1) GO TO 10
C * DUMP THE OUTPUT BUFFER TO BE SURE COMMANDS ARE SENT
        CALL DUMP
C * SIMULATE PROMPT FOR NON-PROMPTING SYSTEMS
C * REMOVE THIS LINE FOR PROMPTING SYSTEMS
        IF (KPRMOD.GT.0) CALL ADEOUT (KPRLEN,KPSTRG)
C * GET LINE OF INPUT FROM TERMINAL
        CALL ADEIN (KINEND,KINBUF)
        KINPT=0
        GO TO 20
C * GET INPUT VIA BLOCK MODE
10      KEOMH=1
        CALL BLOKIO
        KEOMH=0
        KINPT=0
```

```
C * MOVE CHARACTERS FROM INPUT BUFFER TO USER ARRAY
C * EXIT IF REQUEST SATISFIED
20    IF (IREC.GE.IREQ) GO TO 50
C * CHECK FOR EMPTY BUFFER
      IF (KINPT.GE.KINEND) GO TO 30
      IREC=IREC+1
      KINPT=KINPT+1
      ISTRNG(IREC)=KINBUF(KINPT)
      GO TO 20
C * PAD WITH BLANKS
30    JFIRST=IREC+1
      DO 40 I=JFIRST,IREQ
      ISTRNG(I)=32
40    CONTINUE
C * EXIT IF BUFFER NOT EMPTY
50    IF (KINPT.LT.KINEND) RETURN
C * ZERO THE INPUT BUFFER
      KINEND=0
      KINPT=0
      RETURN
      END
C
C----------SUBROUTINE--DUMP-----
C
      SUBROUTINE DUMP
C * DUMP DUMPS THE OUTPUT BUFFER
      DIMENSION IDUMMY(1)
      CALL STOUT (0,IDUMMY)
      RETURN
      END
```

```
C
C----------SUBROUTINE--STOUT----
C
      SUBROUTINE STOUT (ILEN,ISTRNG)
C * STOUT IS THE GENERAL OUTPUT ROUTINE WHICH SHOULD BE USED FOR ALL
C * OUTPUT TO THE 411X TERMINALS
C * IT BUFFERS OUTPUT, BREAKING THE INPUT STRING INTO BUFFER-SIZE
C * CHUNKS IF THE STRING IS TOO LONG
C * ILEN   - NUMBER OF CHARACTERS TO BE TRANSMITTED
C * ISTRNG - ARRAY HOLDING ADE STRING TO BE TRANSMITTED
C * JBSIZE - SIZE OF BUFFER (LINESIZE OR BLOCK LENGTH)
C * JNUM   - NUMBER OF CHARACTERS TO BE BUFFERED 'THIS TIME'
C * JSENT  - NUMBER OF CHARACTERS ALREADY MOVED INTO BUFFER
C * KOUTBF - OUTPUT BUFFER
C * KOUTPT - POINTER TO END OF OUTPUT BUFFER
       COMMON /COMM/ KOUTPT,KOUTBF(512),KINEND,KINPT,KINBUF(512),
      & KRESLU,KIGDEL,KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),
      & KEOMC1,KEOMC2,KCHARS(4),KBUFSZ,KBPLOT,KBPSW,
      & KBARM,KBMODE,KHLENH,KHEADH(10),KHLENT,KHEADT(10),KCONTH,
      & KCONTT,KENDH,KENDT,KNXNOH,KNXMTH(20),KNXNOT,KNXMTT(20),
      & KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,KPACKT,KBLENH,KBLENT,
      & KBLINE,KBLOKH,KEOPH,KEOMH,KEOFH,KEOMT,KEOFT
       DIMENSION ISTRNG(1)
C * SET NUMBER ALREADY SENT TO ZERO
       JSENT=0
C * DUMP BUFFER IF ILEN IS NOT POSITIVE
       IF (ILEN.LE.0) GO TO 40
C * GET NUMBER TO BUFFER 'THIS TIME'
10     JNUM=MINO(ILEN-JSENT,KBUFSZ-KOUTPT)
C * IF BUFFER EMPTY AND IN PLOTTER BLOCK MODE, PUT BLOCK START IN
       IF (KOUTPT.GT.0 .OR. KBPSW.NE.1) GO TO 20
       KOUTPT=3
       KOUTBF(1)=27
       KOUTBF(2)=65
       KOUTBF(3)=40
C * INSERT CHARACTERS INTO BUFFER
20     DO 30 I=1,JNUM
       KOUTPT=KOUTPT+1
       JSENT=JSENT+1
       KOUTBF(KOUTPT)=ISTRNG(JSENT)
30     CONTINUE
C * IF BUFFER NOT FULL, EXIT
       IF (KOUTPT.LT.KBUFSZ) RETURN
C * DUMP THE BUFFER
C * SEE IF IN PLOTTER BLOCK MODE WITH SOMETHING IN BUFFER
40     IF (KBPSW.NE.1 .OR. KOUTPT.EQ.0) GO TO 80
C * COMPUTE CHECKSUM, PUT END-OF-BLOCK SEQUENCE INTO BUFFER
       JCHKSM=173
       DO 50 I=4,KOUTPT
       IF (KOUTBF(I).EQ.22) GO TO 50
       JCHKSM=JCHKSM+KOUTBF(I)
50     CONTINUE
60     IF (JCHKSM.LT.4096) GO TO 70
       JCHKSM=JCHKSM-4095
```

```
      GO TO 60
70    KOUTPT=KOUTPT+1
      KOUTBF(KOUTPT)=27
      KOUTPT=KOUTPT+1
      KOUTBF(KOUTPT)=65
      KOUTPT=KOUTPT+1
      KOUTBF(KOUTPT)=41
      KOUTPT=KOUTPT+1
      CALL KIN2AS (JCHKSM,4,0,JLEN,KOUTBF(KOUTPT))
      KOUTPT=KOUTPT+JLEN
      KOUTBF(KOUTPT)=13
      KEOMH=1
C * BRANCH FOR BLOCK MODE
80    IF (KBMODE.EQ.1) GO TO 90
      IF (KOUTPT.GT.0) CALL ADEOUT (KOUTPT,KOUTBF)
      IF (KBPSW.EQ.1) CALL ADEIN (KINEND,KINBUF)
      GO TO 100
90    CALL BLOKIO
C * IF IN PLOTTER BLOCK MODE, SEE IF POSITIVE ACKNOWLEDGE
100   IF (KBPSW.NE.1) GO TO 120
      IF (KINEND.LT.1 .OR. KINEND.GT.2) GO TO 80
      IF (KINBUF(KINEND).EQ.65) GO TO 110
      JTRY=JTRY+1
      IF (JTRY.LE.3) GO TO 80
110   KEOMH=0
      KINEND=0
      KINPT=0
C * BUFFER EMPTY NOW
120   KOUTPT=0
      KBPSW=KBPLOT
C * DO REST OF STRING IF THERE IS SOME LEFT
      IF (JSENT.LT.ILEN) GO TO 10
      RETURN
      END
```

```
C
C----------SUBROUTINE--BLOKIO---
C
      SUBROUTINE BLOKIO
C * BLOKIO CALLS BLKOUT AND BLKIN TO PERFORM ONE BLOCK 'EXCHANGE'
C * JNACK  - NEGATIVE ACKNOWLEDGE INDICATOR
C * JREPET - REPETITION COUNTER
C * KEOPH  - END-OF-PROTOCOL INDICATOR
C * KBLOKH - BLOCK COUNTER (0 OR 1)
      COMMON /COMM/ KOUTPT,KOUTBF(512),KINEND,KINPT,KINBUF(512),
     & KRESLU,KIGDEL,KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),
     & KEOMC1,KEOMC2,KCHARS(4),KBUFSZ,KBPLOT,KBPSW,
     & KBARM,KBMODE,KHLENH,KHEADH(10),KHLENT,KHEADT(10),KCONTH,
     & KCONTT,KENDH,KENDT,KNXNOH,KNXMTH(20),KNXNOT,KNXMTT(20),
     & KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,KPACKT,KBLENH,KBLENT,
     & KBLINE,KBLOKH,KEOPH,KEOMH,KEOFH,KEOMT,KEOFT
C * INITIALIZE REPETITION COUNTER AND ACKNOWLEDGE FLAG
      JREPET=0
      JNACK=0
C * OUTPUT BLOCK TO TERMINAL
10    CALL BLKOUT (JREPET)
      JREPET=JREPET+1
C * GET BLOCK FROM TERMINAL UNLESS NO-ACK END OF PROTOCOL
      IF (KEOPH.NE.3) CALL BLKIN (JNACK)
C * IF FOURTH REPETITION, ASSUME OKAY
      IF (JREPET.EQ.4) GO TO 30
C * IF NEGATIVE ACKNOWLEDGE, RETRANSMIT
      IF (JNACK.EQ.1) GO TO 10
C * FLIP BLOCK COUNT, ZERO OUTPUT BUFFER
20    KBLOKH=1-KBLOKH
      KOUTPT=0
      RETURN
C * GET OUT OF BLOCK MODE
30    KBMODE=0
      CALL ADEOUT (KOUTPT,KOUTBF)
      KOUTPT=0
      KINEND=0
      RETURN
      END
```

```
C
C----------SUBROUTINE--BLKOUT---
C
      SUBROUTINE BLKOUT (IREPET)
      COMMON /COMM/ KOUTPT,KOUTBF(512),KINEND,KINPT,KINBUF(512),
     & KRESLU,KIGDEL,KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),
     & KEOMC1,KEOMC2,KCHARS(4),KBUFSZ,KBPLOT,KBPSW,
     & KBARM,KBMODE,KHLENH,KHEADH(10),KHLENT,KHEADT(10),KCONTH,
     & KCONTT,KENDH,KENDT,KNXNOH,KNXMTH(20),KNXNOT,KNXMTT(20),
     & KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,KPACKT,KBLENH,KBLENT,
     & KBLINE,KBLOKH,KEOPH,KEOMH,KEOFH,KEOMT,KEOFT
      DIMENSION JLINE(256),JBUF(684),JCTL(4)
      DATA JCTL(2)/0/
      IF (IREPET.GT.0) GO TO 15
C* COMPUTE JCTL(1) BLOCK COUNT, END PROTOCOL,END
C* OF FILE, END OF MESSAGE
      JCTL(1)=KBLOKH+KEOPH+KEOFH*32+KEOMH*64
C * COMPUTE CHECKSUM ON KOUTBF AND THE
C * FIRST TWO CONTROL BYTES.
      CALL CHKSUM (KOUTPT,KOUTBF,JCTL,KBYTEH,JCHK1,JCHK2)
      JCTL(3)=JCHK1
      JCTL(4)=JCHK2
      CALL PACKER (JCTL,JBUFLN,JBUF)
C* PUT BLOCK HEADER IN OUTPUT LINE
      DO 10 I=1,KHLENH
      JLINE(I)=KHEADH(I)
10    CONTINUE
C * BUILD AND SEND LINES TO THE TERMINAL
15    JBUFPT=0
20    JLINPT=KHLENH
C * GET NEXT CHARACTER FROM PACKED BUFFER
30    JBUFPT=JBUFPT+1
C * SEE IF PACKED BUFFER IS EMPTY
      IF (JBUFPT.GT.JBUFLN) GO TO 80
      JCHAR=JBUF(JBUFPT)
C * SUBSTITUTE NON-TRANSMITTABLE CHARACTERS IF NEEDED
      JCNTR=0
40    JCNTR=JCNTR+1
      IF (JCNTR.GT.KNXNOH) GO TO 50
      IF (JCHAR.NE.KNXMTH(JCNTR)) GO TO 40
      JLINPT=JLINPT+1
      JLINE(JLINPT) = KMASTH
      JLINPT=JLINPT+1
      JLINE(JLINPT)=64+JCNTR
      GO TO 60
50    JLINPT=JLINPT+1
      JLINE(JLINPT)=JCHAR
C * TEST FOR END OF PACKED CHARACTERS
60    IF (JBUFPT.GE.JBUFLN) GO TO 80
C * TEST FOR END OF LINE
      IF (JLINPT.GE.KBLINE-2) GO TO 70
```

```
C* GO GET ANOTHER CHAR
      GO TO 30
C * PUT IN CONTINUE CHAR AND SEND LINE TO TERMINAL
70    JLINPT=JLINPT+1
      JLINE(JLINPT)=KCONTH
      CALL ADEOUT (JLINPT,JLINE)
      GO TO 20
C * PUT IN END CHAR AND SEND LINE TO TERMINAL
80    JLINPT=JLINPT+1
      JLINE(JLINPT)=KENDH
      CALL ADEOUT (JLINPT,JLINE)
      RETURN
      END
C
C----------SUBROUTINE--PACKER---
C
      SUBROUTINE PACKER (ICTL,ITNUM,ITBUF)
      COMMON /COMM/ KOUTPT,KOUTBF(512),KINEND,KINPT,KINBUF(512),
     & KRESLU,KIGDEL,KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),
     & KEOMC1,KEOMC2,KCHARS(4),KBUFSZ,KBPLOT,KBPSW,
     & KBARM,KBMODE,KHLENH,KHEADH(10),KHLENT,KHEADT(10),KCONTH,
     & KCONTT,KENDH,KENDT,KNXNOH,KNXMTH(20),KNXNOT,KNXMTT(20),
     & KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,KPACKT,KBLENH,KBLENT,
     & KBLINE,KBLOKH,KEOPH,KEOMH,KEOFH,KEOMT,KEOFT
      DIMENSION ICTL(1),ITBUF(1)
C * FIRST, SEE IF NO PACKING IS NEEDED, IF SO BRANCH TO EASY PART
      IF (KBYTEH.EQ.KPACKH) GO TO 60
C * INITIALIZE VARIABLES
      ITNUM=0
      JRPOW=1
      JREGSR=0
      JFCNT=0
      JCCNT=0
      JOFSET=0
      IF (KPACKH.EQ.64) JOFSET=32
C * GET A BYTE TO PUT INTO REGISTER
10    JFCNT=JFCNT+1
      IF (JFCNT.GT.KOUTPT) GO TO 20
C * GET BYTE FROM 'FROM' BUFFER
      JFBYTE=KOUTBF(JFCNT)
      GO TO 30
C * GET BYTE FROM CONTROL BYTE ARRAY
20    JCCNT=JCCNT+1
      IF (JCCNT.GT.4) GO TO 50
      JFBYTE=ICTL(JCCNT)
C * SHIFT REGISTER BY 'FROM POWER', ADD 'FROM BYTE'
30    JREGSR=JREGSR*KBYTEH+JFBYTE
      JRPOW=JRPOW*KBYTEH
C * SEE IF ENOUGH IN REGISTER, IF NOT GET ANOTHER 'FROM' BYTE
40    IF (JRPOW.LT.KPACKH) GO TO 10
C * COMPUTE 'TO' SHIFT FACTOR
      JSHFT=JRPOW/KPACKH
C * GET 'TO' BYTE
      JTBYTE=JREGSR/JSHFT
C * RESET REGISTER
      JREGSR=JREGSR-JTBYTE*JSHFT
      JRPOW=JSHFT
C * PUT 'TO' BYTE INTO BUFFER
      ITNUM=ITNUM+1
      ITBUF(ITNUM)=JTBYTE+JOFSET
      GO TO 40
```

```
C * ALL DONE WITH DATA, NOW PAD LAST BYTE IF NECCESSARY
50     IF (JRPOW.EQ.1) RETURN
       JREGSR=JREGSR*KPACKH/JRPOW
       JRPOW=KPACKH
       GO TO 40
C * 'EASY PART' - JUST MOVE BYTES STRAIGHT ACROSS
60     IF (KOUTPT.LE.0) GO TO 80
       DO 70 I=1,KOUTPT
       ITBUF(I)=KOUTBF(I)
70     CONTINUE
80     DO 90 I=1,4
       J=KOUTPT+I
       ITBUF(J)=ICTL(I)
90     CONTINUE
       ITNUM=KOUTPT+4
       RETURN
       END
C
C----------SUBROUTINE--BLKIN---
C
       SUBROUTINE BLKIN (INACK)
C * BLKIN GETS A BLOCK FROM THE TERMINAL
       COMMON /COMM/ KOUTPT,KOUTBF(512),KINEND,KINPT,KINBUF(512),
      & KRESLU,KIGDEL,KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),
      & KEOMC1,KEOMC2,KCHARS(4),KBUFSZ,KBFLIM,KBPLOT,
      & KBARM,KBMODE,KHLENH,KHEADH(10),KHLENT,KHEADT(10),KCONTH,
      & KCONTT,KENDH,KENDT,KNXNOH,KNXMTH(20),KNXNOT,KNXMTT(20),
      & KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,KPACKT,KBLENH,KBLENT,
      & KBLINE,KBLOKH,KEOPH,KEOMH,KEOFH,KEOMT,KEOFT
       DIMENSION JBUF(684),JLINE(256),JCTL(4)
       DATA JCTL(2)/0/
C * INITIALIZE NUMBER OF PACKED CHARACTERS RECEIVED
       JNUM=0
C * INITIALIZE MASTER CHARACTER FOUND FLAG
       JMASTR=0
C * GET A LINE OF INPUT FROM THE TERMINAL
C * SIMULATE PROMPT MODE IF TURNED ON
10     IF (KPRMOD.GT.0) CALL ADEOUT (KPRLEN,KPSTRG)
       CALL ADEIN(JLEN,JLINE)
C * IF WRONG LAST CHARACTER, NACK BLOCK
       IF (JLINE(JLEN).NE.KENDT .AND. JLINE(JLEN).NE.KCONTT) GO TO 90
C * SCAN LINE FOR START OF HEADER
       JLINPT=0
20     JLINPT=JLINPT+1
       IF (JLINPT.GE.JLEN) GO TO 90
       IF (JLINE(JLINPT).NE.KHEADT(1)) GO TO 20
C * SCAN FOR REST OF HEADER
       JHEDPT=1
30     JHEDPT=JHEDPT+1
       IF (JHEDPT.GT.KHLENT) GO TO 35
       JLINPT=JLINPT+1
       IF (JLINE(JLINPT).NE.KHEADT(JHEDPT)) GO TO 90
       GO TO 30
```

```
C * FILL TEMP BUFFER, TOTAL IN JNUM, DO NOT INCLUDE LAST CHAR
C * IF MASTER CHAR DETECTED, THEN DO SUBSTITUTION
35     IF (JMASTR.EQ.1) GO TO 50
40     JLINPT=JLINPT+1
       IF (JLINPT.GE.JLEN) GO TO 60
       IF (JLINE(JLINPT).EQ.KMASTT) GO TO 50
       JNUM=JNUM+1
       JBUF(JNUM)=JLINE(JLINPT)
       GO TO 40
50     JMASTR=1
       JLINPT=JLINPT+1
       IF (JLINPT.GE.JLEN) GO TO 10
       JSUB=JLINE(JLINPT)-64
       JNUM=JNUM+1
       JBUF(JNUM)=KNXMTT(JSUB)
       JMASTR=0
       GO TO 40
C * GET ANOTHER LINE IF THIS WAS NOT THE LAST
60     IF (JLINE(JLEN).EQ.KCONTT) GO TO 10
C * UNPACK THE BUFFER, INTO STIN BUFFER IF EOM BIT SET
       IF (KEOMH.NE.1) GO TO 70
       CALL UNPACK (JNUM,JBUF,KINEND,KINBUF,JCTL)
       IF (MOD(JCTL(1),4).NE.KBLOKH+KEOPH) GO TO 90
       CALL CHKSUM (KINEND,KINBUF,JCTL,KBYTET,JCHK1,JCHK2)
C * STRIP FINAL EOM CHAR
       IF (KINBUF(KINEND).EQ.KEOMC1 .OR. KINBUF(KINEND).EQ.KEOMC2)
      &   KINEND=KINEND-1
       GO TO 80
C * IF EOM IS NOT SET, THEN PUT RETURNING EMPTY BLOCK IN JLINE
70     CALL UNPACK (JNUM,JBUF,JLEN,JLINE,JCTL)
       IF (MOD(JCTL(1),4).NE.KBLOKH+KEOPH) GO TO 90
       CALL CHKSUM (JLEN,JLINE,JCTL,KBYTET,JCHK1,JCHK2)
80     IF (JCHK1.NE.JCTL(3) .OR. JCHK2.NE.JCTL(4)) GO TO 90
C * POSITIVE ACKNOWLEDGE
       INACK=0
       KEOMT=JCTL(1)/64
       KEOFT=JCTL(1)/32-KEOMT*2
       RETURN
C * NEGATIVE ACKNOWLEDGE
90     INACK=1
       RETURN
       END
```

```
C
C----------SUBROUTINE--UNPACK---
C
      SUBROUTINE UNPACK (IFNUM,IFBUF,ITNUM,ITBUF,ICTL)
      COMMON /COMM/ KOUTPT,KOUTBF(512),KINEND,KINPT,KINBUF(512),
     & KRESLU,KIGDEL,KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),
     & KEOMC1,KEOMC2,KCHARS(4),KBUFSZ,KBPLOT,KBPSW,
     & KBARM,KBMODE,KHLENH,KHEADH(10),KHLENT,KHEADT(10),KCONTH,
     & KCONTT,KENDH,KENDT,KNXNOH,KNXMTH(20),KNXNOT,KNXMTT(20),
     & KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,KPACKT,KBLENH,KBLENT,
     & KBLINE,KBLOKH,KEOPH,KEOMH,KEOFH,KEOMT,KEOFT
      DIMENSION IFBUF(1),ITBUF(1),ICTL(1)
C * SEE IF PACKING NEEDED, IF NOT BRANCH TO 'EASY WAY'
      IF (KBYTET.EQ.KPACKT) GO TO 70
C * INITIALIZE VARIABLES
      JOVER=0
      ITNUM=0
      JRPOW=1
      JREGSR=0
      JFCNT=0
      JOFSET=0
      IF (KPACKT.EQ.64) JOFSET=32
C * GET A BYTE TO PUT INTO THE REGISTER
10    JFCNT=JFCNT+1
      IF (JFCNT.GT.IFNUM) GO TO 40
C * GET NEXT 'FROM' BYTE
      JFBYTE=IFBUF(JFCNT)-JOFSET
C * SHIFT REGISTER BY 'FROM POWER', ADD 'FROM BYTE'
      JREGSR=JREGSR*KPACKT+JFBYTE
      JRPOW=JRPOW*KPACKT
C * SEE IF ENOUGH IN REGISTER, IF NOT GET ANOTHER 'FROM' BYTE
20    IF (JRPOW.LT.KBYTET) GO TO 10
C * DETERMINE 'TO' SHIFT FACTOR
      JSHFT=JRPOW/KBYTET
C * GET 'TO' CHARACTER FROM REGISTER
      JTBYTE=JREGSR/JSHFT
C * RESET REGISTER
      JREGSR=JREGSR-JTBYTE*JSHFT
      JRPOW=JSHFT
C * PUT 'TO' BYTE INTO SYSTEM INPUT BUFFER OR CONTROL BYTE ARRAY
      IF (ITNUM.LT.KBLENT-4) GO TO 30
C * CONTROL BYTE ARRAY (OVERFLOW CONDITION)
      JOVER=JOVER+1
      ICTL(JOVER)=JTBYTE
      GO TO 20
C * SYSTEM INPUT BUFFER
30    ITNUM=ITNUM+1
      ITBUF(ITNUM)=JTBYTE
      GO TO 20
```

```
C * UNPACKING DONE, SHIFT FINAL CHARS INTO CONTROL BYTES
40    IF (JOVER.EQ.4) RETURN
      JCPT=4
      IF (JOVER.EQ.0) GO TO 60
50    ICTL(JCPT)=ICTL(JOVER)
      JOVER=JOVER-1
      JCPT=JCPT-1
      IF (JOVER.GT.0) GO TO 50
60    IF (JCPT .EQ. 0) RETURN
      ICTL(JCPT)=ITBUF(ITNUM)
      JCPT=JCPT-1
      ITNUM=ITNUM-1
      GO TO 60
C * THE 'EASY WAY'
70    ITNUM=0
      IF (KEOMH.EQ.0) GO TO 90
      ITNUM=IFNUM-4
      IF (ITNUM.LE.0) GO TO 90
      DO 80 I=1,ITNUM
      ITBUF(I)=IFBUF(I)
80    CONTINUE
90    DO 100 I=1,4
      J=ITNUM+I
      ICTL(I)=IFBUF(J)
100   CONTINUE
      RETURN
      END
C
C----------SUBROUTINE--CHKSUM---
C
      SUBROUTINE CHKSUM (ILEN,IARRAY,ICTL,IPOWER,ICHK1,ICHK2)
C * CHKSUM COMPUTES THE CHECKSUM OF THE INPUT ARRAYS
      DIMENSION IARRAY(1),ICTL(1)
C * INITIALIZE CHECKSUM ACCUMULATORS
      MAXBYT=IPOWER-1
      ICHK1=MAXBYT
      ICHK2=MAXBYT
C * DO MAIN ARRAY IF NOT EMPTY
      IF (ILEN.LE.0) GO TO 20
      DO 10 I=1,ILEN
      ICHK1=JADRND(ICHK1,IARRAY(I),MAXBYT)
      ICHK2=JADRND(ICHK2,ICHK1,MAXBYT)
10    CONTINUE
C * DO FIRST TWO CONTROL BYTES
20    DO 30 I=1,2
      ICHK1=JADRND(ICHK1,ICTL(I),MAXBYT)
      ICHK2=JADRND(ICHK2,ICHK1,MAXBYT)
30    CONTINUE
C * RETURN CHECKSUM CHARACTERS
      ICHK1=MAXBYT-ICHK1-ICHK2
      IF (ICHK1.LT.1) ICHK1=ICHK1+MAXBYT
      RETURN
      END
C
C----------FUNCTION--JADRND---
C
      FUNCTION JADRND (INUM1,INUM2,MAXBYT)
      JTEMP=INUM1+INUM2
      IF (JTEMP.GT.MAXBYT) JTEMP=JTEMP-M@XBYT
      JADRND=JTEMP
      RETURN
      END
```

```
C
C----------SUBROUTINE--RELPAK-----
C
      SUBROUTINE RELPAK (RNUM,ILEN,ISTRNG)
C * RELPAK TRANSLATES A REAL NUMBER INTO 411X FORMAT
C * THE MANTISSA IS ALWAYS EITHER 0 OR FROM 2**14 TO 2**15-1
      DIMENSION ISTRNG(1)
C * FIRST BRING THE NUMBER INTO THE VALID RANGE
      SNUM=AMIN1(32767.0,AMAX1(-32767.0,RNUM))
C * SEE IF IT IS NEAR ZERO
      IF (ABS(SNUM).LT..0001) GO TO 10
C * COMPUTE MANTISSA AND EXPONENT USING LOGS
      JEXP=IFIX(ALOG(ABS(SNUM))/ALOG(2.))-14
      JMANT=IFIX(SNUM*FLOAT(2**(-JEXP)))
      GO TO 20
C * NUMBER CONSIDERED TO ZERO
10    JEXP=0
      JMANT=0
C * USE INTEGER PACKING ROUTINE TO TRANSLATE
20    CALL INTPAK (JMANT,L1,ISTRNG(1))
      CALL INTPAK (JEXP,L2,ISTRNG(L1+1))
      ILEN=L1+L2
      RETURN
      END
C
C----------SUBROUTINE--INTRAY--
C
      SUBROUTINE INTRAY (LENINT,INTS,LENADE,IADE)
C * INTRAY TRANSLATES AN INTEGER ARRAY INTO 411X FORMAT
      DIMENSION INTS(1),IADE(1)
      CALL INTPAK (LENINT,LENADE,IADE(1))
      IF (LENINT.LE.0) RETURN
      DO 10 I=1,LENINT
      CALL INTPAK (INTS(I),LEN1,IADE(LENADE+1))
      LENADE=LENADE+LEN1
10    CONTINUE
      RETURN
      END
C
C----------SUBROUTINE--ADERAY--
C
      SUBROUTINE ADERAY (LEN,IRAY,LENRET,JRAY)
C * ADERAY TRANSLATES AN ADE ARRAY INTO 411X FORMAT
      DIMENSION IRAY(1),JRAY(1)
      CALL INTPAK (LEN,LENRET,JRAY(1))
      IF (LEN.LE.0) RETURN
      DO 10 I=1,LEN
      LENRET=LENRET+1
      JRAY(LENRET)=IRAY(I)
10    CONTINUE
      RETURN
      END
```

```
C
C----------SUBROUTINE--ADERAY--
C
      SUBROUTINE ADERAY (LEN,IRAY,LENRET,JRAY)
C * ADERAY TRANSLATES AN ADE ARRAY INTO 411X FORMAT
      DIMENSION IRAY(1),JRAY(1)
      CALL INTPAK (LEN,LENRET,JRAY(1))
      IF (LEN.LE.0) RETURN
      DO 10 I=1,LEN
      LENRET=LENRET+1
      JRAY(LENRET)=IRAY(I)
10    CONTINUE
      RETURN
      END
C
C----------SUBROUTINE--INTPAK---
C
      SUBROUTINE INTPAK (INT,ILEN,ISTRNG)
C * INTPAK TRANSLATES AN INTEGER INTO 411X FORMAT
      COMMON /COMM/ KOUTPT,KOUTBF(512),KINEND,KINPT,KINBUF(512),
     & KRESLU,KIGDEL,KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),
     & KEOMC1,KEOMC2,KCHARS(4),KBUFSZ,KBPLOT,KBPSW,
     & KBARM,KBMODE,KHLENH,KHEADH(10),KHLENT,KHEADT(10),KCONTH,
     & KCONTT,KENDH,KENDT,KNXNOH,KNXMTH(20),KNXNOT,KNXMTT(20),
     & KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,KPACKT,KBLENH,KBLENT,
     & KBLINE,KBLOKH,KEOPH,KEOMH,KEOFH,KEOMT,KEOFT
      DIMENSION ISTRNG(1)
C * INTIALIZE LENGTH OF ARRAY RETURNED
      ILEN=0
C * FIRST BRING NUMBER INTO VALID RANGE
      JINT=MIN0(65535,IABS(INT))
C * COMPUTE THE TWO HI-I'S AND THE LO-I
      JHI1=JINT/1024+64
      JHI2=MOD(JINT/16,64)+64
      JLOI=MOD(JINT,16)+32
      IF (INT.GE.0) JLOI=JLOI+16
C * SEE IF HI-I'S NEEDED
      IF (JHI1.NE.64) GO TO 10
      IF (JHI2.NE.64) GO TO 30
      GO TO 50
C * INSERT FIRST HI-I (EXPANDING TO <ESC><?> IF NEEDED)
10    IF (JHI1.EQ.127 .AND. KIGDEL.EQ.1) GO TO 20
      ILEN=1
      ISTRNG(1)=JHI1
      GO TO 30
20    ILEN=2
      ISTRNG(1)=27
      ISTRNG(2)=63
C * INSERT SECOND HI-I (EXPANDING TO <ESC><?> IF NEEDED)
30    IF (JHI2.EQ.127 .AND. KIGDEL.EQ.1) GO TO 40
      ILEN=ILEN+1
      ISTRNG(ILEN)=JHI2
      GO TO 50
40    ILEN=ILEN+2
      ISTRNG(ILEN-1)=27
      ISTRNG(ILEN)=63
C * INSERT LO-I
50    ILEN=ILEN+1
      ISTRNG(ILEN)=JLOI
      RETURN
      END
```

```
C
C----------SUBROUTINE--XYTRN----
C
      SUBROUTINE XYTRN (IX,IY,LEN,ICHARS)
C * XYTRN TRANSLATES X-Y COORDINATES INTO OPTIMIZED ADE STRING
C * IX,IY  - X,Y COORDINATES TO BE TRANSLATED
C * LEN    - LENGTH OF OPTIMIZED STRING <=7
C * ICHARS - OPTIMIZED STRING ARRAY
C * JCHARS - LOCAL TRANSLATION ARRAY
C * KCHARS - COMMON ARRAY TO REMEMBER LAST TRANSLATION (EXCEPT LOX)
C * KRESLU - RESOLUTION OF TRANSLATION [10,12]
C * KIGDEL - IGNORE DEL (ADE 127) FLAG
      COMMON /COMM/ KOUTPT,KOUTBF(512),KINEND,KINPT,KINBUF(512),
     & KRESLU,KIGDEL,KPRMOD,KPRLEN,KPSTRG(10),KEOFLN,KEOFST(10),
     & KEOMC1,KEOMC2,KCHARS(4),KBUFSZ,KBPLOT,KBPSW,
     & KBARM,KBMODE,KHLENH,KHEADH(10),KHLENT,KHEADT(10),KCONTH,
     & KCONTT,KENDH,KENDT,KNXNOH,KNXMTH(20),KNXNOT,KNXMTT(20),
     & KMASTH,KMASTT,KBYTEH,KBYTET,KPACKH,KPACKT,KBLENH,KBLENT,
     & KBLINE,KBLOKH,KEOPH,KEOMH,KEOFH,KEOMT,KEOFT
      DIMENSION ICHARS(1),JCHARS(5)
C * FIRST BRING COORDINATES INTO VALID RANGE
      JX=MIN0(4095,MAX0(0,IX))
      JY=MIN0(4095,MAX0(0,IY))
C * CALCULATE 10-BIT RESOLUTION CHARACTERS
      JCHARS(1)=JY/128+32
      JCHARS(3)=JY/4-JY/128*32+96
      JCHARS(4)=JX/128+32
      JCHARS(5)=JX/4-JX/128*32+64
C * INITIALIZE ARRAY LENGTH
      LEN=0
C * SEE IF HI-Y NEEDED
      IF (JCHARS(1).EQ.KCHARS(1)) GO TO 10
C * INSERT HI-Y
      LEN=1
      ICHARS(1)=JCHARS(1)
      KCHARS(1)=JCHARS(1)
C * SEE IF 12-BIT RESOLUTION
10    IF (KRESLU.NE.12) GO TO 20
C * COMPUTE EXTRA-LO-Y
      JCHARS(2)=(JY-JY/4*4)*4+(JX-JX/4*4)+96
C * SEE IF EXTRA-LO-Y NEEDED
      IF (JCHARS(2).EQ.KCHARS(2)) GO TO 20
C * INSERT EXTRA-LO-Y
      LEN=LEN+1
      ICHARS(LEN)=JCHARS(2)
      KCHARS(2)=JCHARS(2)
      GO TO 30
```

```
C * SEE IF LO-Y NEEDED
20    IF (JCHARS(3).NE.KCHARS(3)) GO TO 30
      IF (JCHARS(4).EQ.KCHARS(4)) GO TO 50
C * INSERT LO-Y
30    LEN=LEN+1
      ICHARS(LEN)=JCHARS(3)
      KCHARS(3)=JCHARS(3)
C * EXPAND LO-Y TO <ESC><?> IF NECESSARY
      IF (JCHARS(3).NE.127 .OR. KIGDEL.EQ.0) GO TO 40
      ICHARS(LEN)=27
      LEN=LEN+1
      ICHARS(LEN)=63
C * SEE IF HI-X NEEDED
40    IF (JCHARS(4).EQ.KCHARS(4)) GO TO 50
      LEN=LEN+1
      ICHARS(LEN)=JCHARS(4)
      KCHARS(4)=JCHARS(4)
C * ALWAYS INCLUDE LO-X
50    LEN=LEN+1
      ICHARS(LEN)=JCHARS(5)
      RETURN
      END
C
C----------SUBROUTINE--RELUNP---
C
      SUBROUTINE RELUNP (ISTRNG,RELPAR)
C * RELUNP UNPACKS A REAL-REPORT
      DIMENSION ISTRNG(1)
      CALL INTUNP (ISTRNG,JMANT)
      CALL INTUNP (ISTRNG(4),JEXP)
      RELPAR=JMANT*2*JEXP
      RETURN
      END
C
C----------SUBROUTINE--INTUNP---
C
      SUBROUTINE INTUNP (ISTRNG,INTPAR)
C * INTUNP UNPACKS AN INT-REPORT
      DIMENSION ISTRNG(1)
      INTPAR=(ISTRNG(1)-32)*1024+(ISTRNG(2)-32)*16+MOD(ISTRNG(3),16)
      IF (ISTRNG(3).LT.48) INTPAR=-INTPAR
      RETURN
      END
C
C----------SUBROUTINE--XYUNP---
C
      SUBROUTINE XYUNP (IRAY,IX,IY)
C * XYUNP UNPACKS TERMINAL-TO-HOST X-Y FORMAT
      DIMENSION IRAY(1)
      IX=(IRAY(4)-32)*128+(IRAY(5)-32)*4+IRAY(2)-IRAY(2)/4*4
      IY=(IRAY(1)-32)*128+(IRAY(3)-32)*4 +(IRAY(2)-32)/4
      RETURN
      END
```

```
C
C----------SUBROUTINE--KAS2IN--
C
      SUBROUTINE KAS2IN (LEN,IADE,INT)
C
C * TRANSLATES ADE CHARACTERS TO THEIR INTEGER VALUE
C * LEN  - NUMBER OF CONSECUTIVE ADE CHARACTERS
C * IADE - ADE CHARACTERS TO BE TRANSLATED
C * INT  - INTEGER VALUE AFTER TRANSLATION
C
      DIMENSION IADE(1)
      INT=0
      J=LEN+1
      POWER=.1
10    J=J-1
      IF (J.LE.0) GO TO 30
      ICHAR=IADE(J)
      IF (ICHAR.EQ.45) GO TO 20
      IF (ICHAR.LT.48 .OR. ICHAR.GT.57) GO TO 10
      POWER=POWER*10.
      INT=INT+(ICHAR-48)*IFIX(POWER)
      GO TO 10
20    INT=-INT
30    RETURN
      END
```

```
C
C----------SUBROUTINE--KIN2AS--
C
      SUBROUTINE KIN2AS (INT,LENLIM,IFILL,LENRET,IADE)
C
C * TRANSLATES INTEGER VALUES INTO ADE CHARACTERS
C * INT    - INTEGER VALUE TO BE TRANSLATED
C * LENLIM - LIMIT ON NUMBER OF CHARACTERS
C * IFILL  - FILL CHARACTER
C *             >0 RIGHT JUSTIFY, FILL WITH IFILL
C *             =0 LEFT JUSTIFY, NO FILL
C *             <0 LEFT JUSTIFY, FILL WITH ABS(IFILL)
C * LENRET - NUMBER OF CHARACTERS RETURNED
C * IADE   - ADE CHARACTER ARRAY
C
      DIMENSION IADE(1),JADE(16)
C * SET UP WORKING VARIABLES
      JNT=IABS(INT)
      LENRET=0
C * TRANSLATE INTEGER INTO WORK ARRAY
10    LENRET=LENRET+1
      IF (LENRET.GT.LENLIM) GO TO 100
      JADE(LENRET)=JNT-JNT/10*10+48
      JNT=JNT/10
      IF (JNT.NE.0) GO TO 10
C * ADD MINUS SIGN IF NEGATIVE
      IF (INT.GE.0) GO TO 20
      LENRET=LENRET+1
      IF (LENRET.GT.LENLIM) GO TO 100
      JADE(LENRET)=45
C * LEFT JUSTIFIED OR NO FILL
20    IF (IFILL.GT.0) GO TO 50
      DO 30 I=1,LENRET
      J=LENRET-I+1
      IADE(I)=JADE(J)
30    CONTINUE
      IF (IFILL.EQ.0) RETURN
```

```
C * LEFT JUSTIFIED
      IF (LENRET.EQ.LENLIM) RETURN
      J=LENRET+1
      DO 40 I=J,LENLIM
      IADE(I)=-IFILL
40    CONTINUE
      RETURN
C * RIGHT JUSTIFIED
50    J=LENLIM-LENRET
      IF (J.EQ.0) GO TO 70
      DO 60 I=1,J
      IADE(I)=IFILL
60    CONTINUE
70    K=J+1
      DO 80 I=K,LENLIM
      J=LENRET-I+K
      IADE(I)=JADE(J)
80    CONTINUE
      RETURN
C * OVERFLOW - FILL WITH ASTERISKS
100   DO 110 I=1,LENLIM
      IADE(I)=42
110   CONTINUE
      LENRET=LENLIM
      RETURN
      END
```

Appendix D

COLOR COORDINATES

At any one time, the 4113 can display only a limited number of different color mixtures. However, you can select which color mixtures are in use from a universe of 4096 possible color mixtures. To do this, you use the <set-background-color> and <set-surface-color-map> commands. These commands are described in Section 7 and in the 4110 Series Command Reference Manual.

In these commands, you specify a particular color mixture using one of three color coordinate systems: RGB (Red, Green, Blue), CMY (Cyan, Magenta, Yellow), and HLS (Hue, Lightness, Saturation). This appendix describes those color coordinate systems.

<SET-COLOR-MODE> COMMAND

On power-up, the 4113 is set to use the HLS system. You can select other coordinate systems with the <set-color-mode> command. (The SETUP mode name for this command is CMODE.)

<set-color-mode>

= (ESC)(T)(M)
<int: *color-specifying-mode*>
<int: *color-overlay-mode*>
<int: *gray-mode*> .

The first parameter, color-specifying-mode, is 1 to select the RGB color coordinate system, 2 to select the CMY system, or 3 to select the HLS system. If this parameter is zero, the color-specifying-mode is left unchanged.

(For information about the other two parameters, see the descriptions of the <set-color-mode> command in Section 7 of this manual, and in the 4110 Series Command Reference manual.)

In SETUP mode, the operator types the keywords "RGB," "CMY," and "HLS" instead of the numbers "1," "2," and "3." Consider, for instance, the following examples:

CMODE RGB	Selects the RGB color coordinate system.
CMODE CMY	Selects the CMY color coordinate system.
CMODE HLS	Selects the HLS color coordinate system.
CMODE 0 OPAQUE	Leaves the color coordinate system unchanged, but sets the overlay mode to OPAQUE.

RGB COORDINATE SYSTEM

In the RGB color coordinate system, you specify a color mixture as percentages of red, green, and blue, in that order. Each color coordinate is an integer in the range from 0 to 100.

For instance, one way to set the background color to red is to issue these commands:

<set-color-mode: 1, 0, 1>	Select the RGB color coordinate system. Leave the overlay mode unchanged, but set the gray mode to "COL" to ensure that the display is in color rather than in black and white.
<set-background-color: 100,0,0>	Set the intensities of the red, green, and blue electron beams to 100%, 0%, and 0% of their maximum values, respectively.

Likewise, you can set color-index one on surface number three to "green," as follows:

<set-color-mode: 1, 0, 1>	Select RGB color coordinates.
<set-surface-color-map: 1, (1,0,100,0)>	Set the color mixture for surface one, color-index one, as follows: 0% red, 100% green, 0% blue.

In SETUP mode, these latter two commands are typed as follows:

```
CMODE  RGB
CMAP  1  1,0,100,0
```

From the host computer, these same commands are sent as escape sequences:

<set-color-mode: 1,0,1>

= (ESC)(T)(M)<int: 1><int: 0><int: 1>

= (ESC)(T)(M)(1)(0)(1) .

<set-surface-color-map: 1, (1,0,100,0)>

= (ESC)(T)(G)
<int: 1>
<int-array: (1,0,100,0)>

= (ESC)(T)(G)
<int: 1>
<int: 4><int: 1><int: 0>
<int: 100><int: 0>

= (ESC)(T)(G)(1)(4)(1)(0)(F)(4)(0) .

CMY COORDINATE SYSTEM

In the CMY system, the three color coordinates are percentages of cyan, magenta, and yellow pigments. Each coordinate is an integer in the range from 0 to 100.

(The additive primaries—red, green, and blue—are used when mixing lights to produce color mixtures. The subtractive primaries—cyan, magenta, and yellow—are used when mixing pigments.)

The CMY coordinates are related to the RGB coordinates as follows:

C = 100 − R
M = 100 − G
Y = 100 − B

For instance, you can use the following commands to select a red background color:

<set-color-mode: 2, 0, 1>	Select the CMY coordinate system, while leaving the overlay mode unchanged and setting the gray mode to "color" rather than "black and white."
<set-background-color: 0,100,100>	Mix pigments of 0% cyan, 100% magenta, and 100% yellow to produce a "red" color mixture.

In SETUP mode, the operator can type these commands as follows:

```
CMODE  CMY
EC TB  0,100,100
```

(There is no SETUP mode name for the <set-background-color> command, so in SETUP mode the operator must use the escape-sequence op code for that command.)

The same two commands can be sent from the host computer as escape sequences:

<set-color-mode: 2,0,1>

= (ESC)(T)(M)<int: 2><int: 0><int: 1>

= (ESC)(T)(M)(2)(0)(1) .

<set-background-color: 0,100,100>

= (ESC)(T)(B)<int: 0><int: 100><int: 100>

= (ESC)(T)(B)(0)(F)(4)(F)(4) .

HLS COORDINATE SYSTEM

In the HLS coordinate system, the universe of possible color mixtures is represented as a double-ended cone (Figure D-1). The three coordinates are H (hue), L (lightness), and S (saturation).

Hue. The hue coordinate runs around the cone, from 0 to 360 degrees:

Hue Coordinate	Color Name
0	Blue
60	Magenta
120	Red
150	Orange (red-yellow)
180	Yellow
240	Green
300	Cyan

Lightness. The lightness coordinate runs up the cone, from black at the bottom (0% lightness) to white at the top (100% lightness).

Saturation. The saturation coordinate expresses the degree to which a color mixture differs from a shade of gray. This coordinate runs radially outward from the axis of the HLS cone. It is expressed as a percentage of the maximum saturation that is possible at a given lightness level. The most fully saturated color mixtures are at the 50% lightness level, where the double-ended cone is widest.

The HLS coordinate system is easier to use than the RGB or CMY systems; for this reason, it is the default color coordinate system and is in effect when the terminal is turned on.

In the HLS coordinate system, all "red" color mixtures have the same hue angle. For instance, "dark red," "fully saturated red," and "light red" differ only in the lightness coordinate:

Color Name	H	L	S
Dark Red	120	33	100
Fully Saturated Red	120	50	100
Light Red	120	67	100

Likewise, you can get light-colored mixtures of different hues by setting the lightness coordinate to a relatively large value and varying only the hue coordinate:

Color Name	H	L	S
Light Red	120	67	100
Light Orange	150	67	100
Light Yellow	180	67	100
Light Green	240	67	100
Light Blue	0	67	100

Again, different mixtures of a given color with gray can be achieved by varying only the saturation coordinate:

Color Name	H	L	S
50% Gray	120	50	0
Grayish Red	120	50	50
Fully Saturated Red	120	50	100

For instance, to set color index 2 on Surface 1 to a light shade of green, you could issue these commands:

<set-color-mode: 3,0,1>	Select the HLS color coordinate system, while leaving the overlay mode unchanged and setting the gray mode so that the display is in color rather than in black and white.
<set-surface-color-map: 1, (2,240,67,100)>	Set the color mixture for Surface 1, color index 2, as follows: a green hue (H= 240), of a light shade (L= 67), with the maximum saturation possible at that lightness level (S= 100).

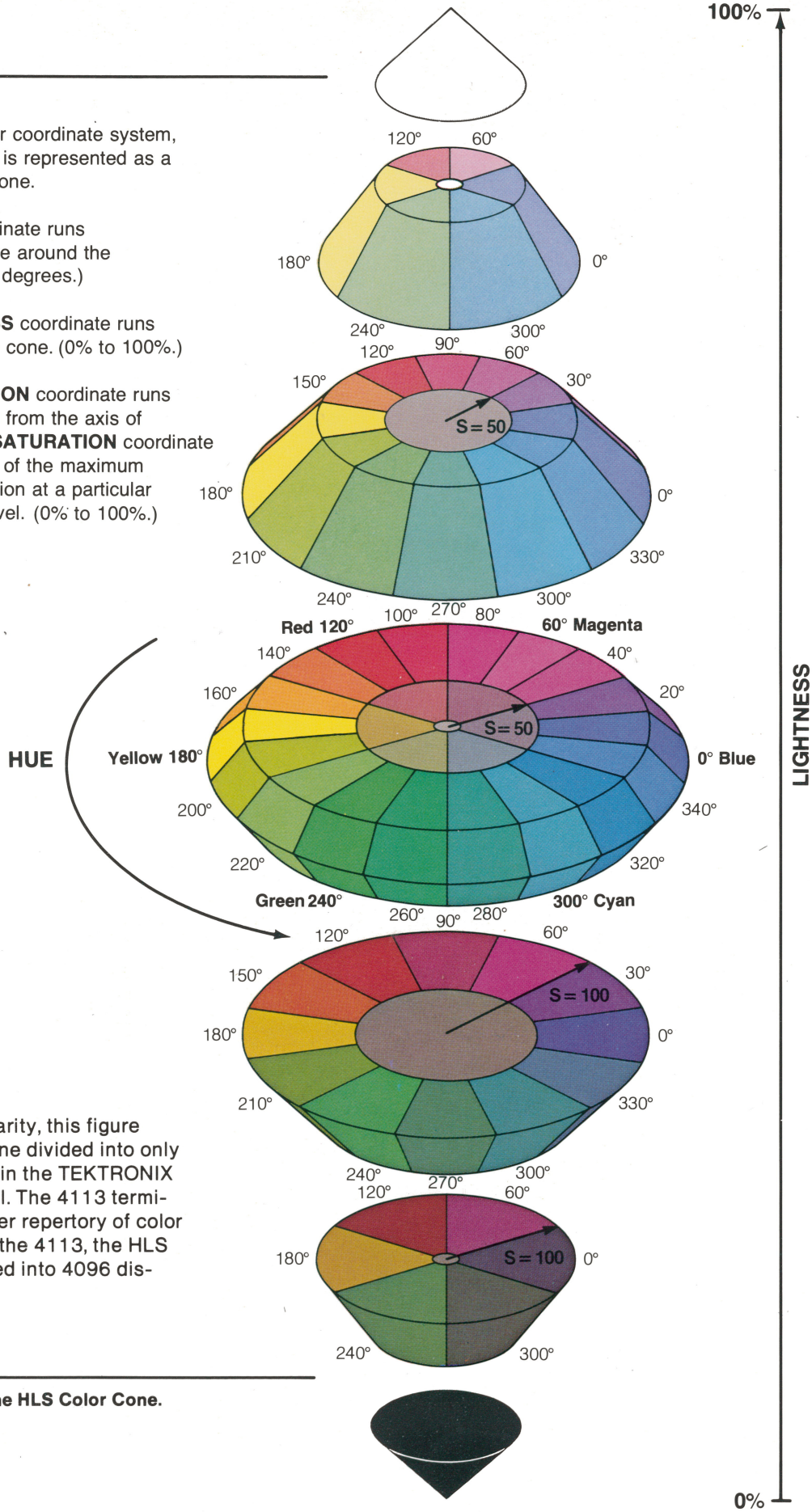
For more information on the <set-color-mode> and <set-surface-color-map> commands, see their descriptions in Section 7 of this manual and in the 4110 Series Command Reference Manual.

In the **HLS** color coordinate system, the color space is represented as a double-ended cone.

The **HUE** coordinate runs counterclockwise around the cone. (0 to 360 degrees.)

The **LIGHTNESS** coordinate runs vertically up the cone. (0% to 100%.)

The **SATURATION** coordinate runs radially outward from the axis of the cone. The **SATURATION** coordinate is a percentage of the maximum possible saturation at a particular **LIGHTNESS** level. (0% to 100%.)



NOTE: For clarity, this figure shows the cone divided into only 64 colors, as in the TEKTRONIX 4027 terminal. The 4113 terminal has a wider repertory of color mixtures; for the 4113, the HLS cone is divided into 4096 distinct cells.

Figure D-1. The HLS Color Cone.