

# **Quick C9**

**Rev. 3.1**



# **Quick C9**

**Manual P/N: 15076**

**Manual Revision: 3.1**

**Manual Revision Date: July 1998**

**Firmware Version: 2.07**

**Copyright 1998  
Deeco Systems  
Division of Computer Dynamics**

**Deeco Systems  
Division of Computer Dynamics  
7640 Pelham Road  
Greenville, SC 29615  
803-627-8800**



All rights reserved. Deeco and SealTouch are registered trademarks of Deeco Systems, a Division of Computer Dynamics, a subsidiary of Total Control Products. All other trademarks are the property of their respective owners. Information furnished by Deeco Systems is believed to be accurate and reliable. However, no responsibility is assumed by Deeco Systems for its use, nor for any infringements of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent rights of Deeco Systems.

Printed in the U.S.A.



# TABLE OF CONTENTS

<b>1.0 INTRODUCTION .....</b>	<b>1</b>
1.1 SealTouch Touch Sensor .....	1
1.2 Text Terminal Operation.....	1
1.3 Graphic Terminal Operation.....	2
<b>2.0 COMMAND DESCRIPTION.....</b>	<b>3</b>
2.1 ANSI 3.64 Description .....	3
2.1.1 Escape Sequences .....	4
2.1.2 Command String Initiator.....	5
2.1.3 Device Command Strings .....	5
2.2 Definitions .....	5
2.3 Error Codes .....	6
<b>3.0 GETTING STARTED WITH C9 QUICK MODE TERMINAL.....</b>	<b>9</b>
3.1 Power-On and Self-Test.....	10
3.2 Testing the Touch System .....	11
3.3 Setting the Terminal to Factory Default Configuration.....	12
3.4 PC Serial Communications Software Selection .....	13
3.5 PC Cabling .....	14
3.6 Using A Windows 95 Computer Without A Mouse.....	15
3.7 PC Configuration & Communication .....	16
3.8 Sending Commands to the Touch Terminal .....	17
3.9 Creating Command Files and Sending Files to the Terminal.....	18
3.10 Sending More Files to the Terminal.....	19
3.11 More About the "Quick C9 Utilities & Demos Disk" .....	20
3.12 Quick Assist .....	20
3.13 Practical Hints .....	20
<b>4.0 BUTTONS .....</b>	<b>21</b>
4.1 Utilities Demos And Batch Files.....	21
4.2 Buttons And Windows.....	22
4.3 Button Structure .....	22
4.4 Scrolling Windows.....	24
4.5 Logos.....	26
4.6 Button Bitmap Labels.....	27
4.7 Window (Button) Layering And Updating .....	28
4.8 Button Pages.....	30
4.9 Multi-State Buttons .....	31
4.10 Multi-State Button Example Walk Through .....	32
4.11 Limitations And Restrictions Of Multi-State Buttons.....	33
4.12 Button Zero (0) .....	33
4.13 Text And Graphics Video Mode .....	33
4.14 Copy Button Command .....	34
4.15 Erase Screen / Clear Active Button .....	35
4.16 Limitations Restrictions And Characteristics.....	35

<b>5.0 MENUS, ORGANIZATION AND ATTRIBUTES .....</b>	<b>37</b>
5.1 Menu Creation Sequence .....	37
<b>6.0 KEYBOARDS .....</b>	<b>39</b>
6.1 Static Keyboards.....	39
6.2 Keyboard Styles.....	39
6.3 Creating And Modifying A Keyboard .....	40
6.4 Keyboard Hotkey.....	41
6.5 Access To Keyboard And Setup Screen.....	41
6.6 Drawing Keyboards .....	41
<b>7.0 TOUCH SYSTEM OPERATION .....</b>	<b>43</b>
7.1 Touch Modes .....	43
7.2 Touch Reports .....	44
7.3 Touch Inquiry.....	44
7.4 Bad-Beam Notification.....	45
<b>8.0 GRAPHICS COMMANDS &amp; TECHNIQUES .....</b>	<b>47</b>
8.1 User-Definable Line Style And Fill Pattern .....	49
8.2 Graphics Clipping.....	50
8.3 Erasing Graphics .....	51
8.4 Graphic Text.....	51
8.5 Fast Vector Mode .....	52
8.6 Auxiliary Memory And Display Lists.....	54
8.6.1 Other Auxiliary Memory List Types .....	55
8.7 User-Definable Fonts .....	55
8.7.1 Font Lists .....	55
8.7.2 Auto-loading character sets.....	56
8.7.3 Defining User-defined characters.....	56
8.8 Drawing Bitmap Images.....	58
8.8.1 Tiling regions with Bitmap Lists .....	59
8.9 Graphic Lists.....	60
8.9.1 Linking A Graphic List To A Button Page.....	62
8.9.2 Alternatives To Graphic Lists .....	62
<b>9.0 GRAPHIC OBJECTS.....</b>	<b>63</b>
9.1 Passive Graphic Objects .....	63
9.2 Active Graphic Objects.....	64
9.3 Working with Graphic Objects .....	66
9.4 Working with Trend Graphs .....	67
9.5 Linking Trend Graphs.....	67
9.6 Water Marks and Control Responses .....	67
9.7 Graphic Object Report.....	67
<b>10.0 MISCELLANEOUS TOPICS.....</b>	<b>69</b>
10.1 Adjusting Touch Screen Sensitivity .....	69
10.2 Self-Check Functions .....	69
10.2.1 System Operation Commands .....	69
10.2.2 Configuration Inquiry .....	70
10.2.3 List-reporting Commands .....	71

10.3 The Real-Time Clock.....	71
10.4 Auto-Configure.....	71
10.5 Set-Reset Functions .....	72
10.6 Periodic Functions.....	72
<b>APPENDIX A: C9 HOST COMMAND REFERENCE.....</b>	<b>73</b>
A.1 C9 Host Command Reference, by category .....	73
A.1.1 Button Functions .....	73
A.1.2 Menu Commands .....	74
A.1.3 Keyboard Functions .....	74
A.1.4 Graphics Drawing Functions .....	74
A.1.5 Graphic List Functions .....	75
A.1.6 Graphic Objects .....	76
A.1.7 Touch Functions.....	76
A.1.8 Screen Functions.....	76
A.1.9 Alpha Cursor Movement Commands.....	77
A.1.10 Printer Functions .....	78
A.1.11 Auxiliary Memory List Functions .....	78
A.1.12 Miscellaneous Functions.....	78
A.1.13 Error Reporting Functions .....	79
A.1.14 Real-Time Clock (RTC) Commands.....	79
A.2 C9 Host Command Reference, Alphabetical listing.....	81
A.3 C9 HOST COMMAND REFERENCE, by command code.....	117
<b>APPENDIX B: COMMAND ERROR REPORTING .....</b>	<b>123</b>
B.1 The Error Reporting Mode command.....	123
<b>B.2 ERROR REPORT FORMATS.....</b>	<b>123</b>
<b>APPENDIX C: ASCII CHARACTERS AND EQUIVALENTS.....</b>	<b>129</b>
<b>INDEX.....</b>	<b>133</b>
<b>WARRANTY.....</b>	<b>139</b>



## 1.0 Introduction

The C9 SealTouch infrared touch sensor provides a user-friendly input device. The flexibility and simplicity of this type of input device makes it the most attractive and least intimidating of all general purpose computer input devices.

### 1.1 SealTouch Touch Sensor

The SealTouch infrared touch sensor has three basic methods of reporting user touches. These methods are described below. These methods support several modes of operation. The message can be sent immediately upon touching the region (entry mode), or only after the finger leaves the screen (exit mode). The exit mode is used with button highlighting which allows accurate selection from tightly spaced regions.

The first method simply reports the X-Y coordinates of the center of the user's finger. There are four independent modes within this method. These allow reporting of entry, exit, tracking, and multiple (error) touches.

The second method, user defined interface objects, detects touches within any rectangular region on the screen and reports back a user-defined message. A very useful (default) mode automatically draws a boundary around the region on the screen and labels this "button" with the message to be sent or, optionally, with a different label. To facilitate rapid menu changes, the sensor can support multiple pages of button definitions and can switch among them with a single command.

The third method of operation allows the program to overlay the screen with a standard keyboard layout, and to report touches as if they were typed on a keyboard. This allows a user to enter specific information, names, passwords, etc., without having to have a separate keyboard available. This keyboard can be made to appear under software control or, if the software permits, under user control.

It is possible to use all three of these methods concurrently by enabling those desired. There is a hierarchy which causes the on-screen keyboard to override the user-defined-buttons, which in turn overrides coordinate reporting.

### 1.2 Text Terminal Operation

Text terminal operation simulates a DEC style ANSI 3.64 terminal. The terminal can display 2,000 characters arranged as 25 rows of 80 columns. New data sent to the terminal is displayed at the cursor location (a particular row and column), and may be manipulated on the screen by several commands.

### 1.3 Graphic Terminal Operation

The graphic terminal portion of the C9 uses an extension of the ANSI 3.64 standard. Graphic commands will draw vectors, circles, rectangles, and much more.

A graphic object is positioned within raster memory by specifying the coordinates of the object. The basic coordinate of the system, termed the "global coordinate system", assumes that raster memory is one large X-Y plane.

The dimensions of the global coordinate system are 640 pixels wide by 480 scanlines, with scanline 0 at the bottom and scanline 479 at the top.

Within this global system, a local coordinate system may be defined. This local coordinate system, known as the graphic clipping window, is defined by specifying a rectangular region relative to the lower-left corner of the screen. All graphic objects which are drawn on the screen will be "clipped" to this window. The clipping rectangle may be of any size, and by default is the full graphics screen. Graphics commands and techniques are discussed in more detail later in this manual.

## 2.0 Command Description

The commands described in this section fall into two broad categories. The first are commands that the controller has in common with VT100-type terminals. The second category contains commands used to control the special features of the C9.

VT100/VT220 commands control typical alphanumeric terminal features such as cursor positioning, scrolling, and character attributes. The C9 supports many of the commands found in these popular terminals.

The C9 special commands control features that are not found in a typical alphanumeric terminal. There are commands to draw graphic objects, control the raster (screen image) memory, and control optional features such as "Display List Memory."

All commands are encoded in the ANSI 3.64 standard. The best known example of a device using ANSI 3.64 commands is the VT100 terminal. The standard simply defines a way to pass information and commands to terminal-display devices.

The command descriptions which are provided in the Command Reference are listed first in a brief summary, then in a second, more detailed description in which the command and its parameters are defined and explained.

### 2.1 ANSI 3.64 Description

The ANSI 3.64 standard is a method of encoding information and commands for computer peripherals and other devices. The encoding is based on the 7-bit ASCII encoding defined by the ANSI standard 3.4 and the 8-bit extension to that encoding which defines the ANSI 3.41 standard.

A typical ANSI 3.64 command uses a "sequence introducer," "parameters" and a "final character" or "terminator," all encoded in ASCII. The sequence introducer signals the receiving device that a command follows. The parameters are usually decimal numbers required to perform the command. The final character determines which command the receiving device is to perform. A terminator is used when device-specific commands are used.

The C9 uses the ESC, CSI and DCS as sequence introducers and ST as a string terminator. These ANSI-defined symbols are encoded as one to two characters. The two character versions must be used if 7-bit ASCII encoding is used between the host and the receiving device. The symbols are encoded as follows:

ANSI SYMBOL	8-BIT ENCODING	7-BIT ENCODING	ASCII EQUIVALENT
ESC	1BH	1BH	ESC
CSI	9BH	1BH 5BH	ESC [
DCS	9OH	1BH 5OH	ESC P
ST	9CH	1BH 5CH	ESC \
--	98H	1BH 5BH 3CH	ESC [<
--	99H	1BH 5BH 3EH	ESC [>

Table 2-1 ANSI 3.64 Commands

Parameters are ASCII encoded decimal numbers. The digits 0 through 9 are encoded as 30H through 39H in ASCII. Multiple parameters are separated by semi-colons (ASCII code 3BH). The C9 controller extends the 3.64 to allow the specification of negative numbers.

Final characters have ASCII codes which fall between 40H and 7FH. Typically, a final character will be an alphabetic character, either upper or lower case.

Supplemental characters may appear after the introducer and before the first parameter which can alter the operation of the command. The C9 uses such supplemental characters to control non-ANSI features.

An example of an ANSI command sequence is the "cursor position command" used to move the alpha cursor to a particular row and column. The common definition of the command is:

**CSI Pr ; Pc H**

Where **Pr** stands for the decimal number of the row, and **Pc** stands for the decimal number of the column. The parameters are separated by a semi-colon. The command is introduced with the CSI introducer. It is terminated by the final character "H." The spaces shown in the definition are not part of the command, they are used to help illustrate the constituent parts of the command. This form of command definition is used throughout the manual to illustrate the general form of a command.

A specific instance of the "cursor position command" would replace the parameter symbols "Pr" and "Pc" with actual values. For example, to move the cursor to row 12, column 49, the following command form would be used:

**CSI 12 ; 49 H**

The actual information sent from the host to the display controller is the ASCII encoded command string.

In the above example, the ASCII code for each character (except the spaces which are used for illustration) is sent to the controller:

COMMAND STRING CHARACTER/SYMBOL	ASCII CODE
CSI	9BH (8-bit) or 1BH 5BH (7-bit)
1	31H
2	32H
;	3BH
4	34H
9	39H
H	48H

**Table 2-2 ASCII Encoded Command String**

### 2.1.1 Escape Sequences

Escape sequences begin with the ASCII character ESC (1BH), followed by one or more ASCII characters. Escape sequences use only 7-bit characters, and can be used in 7-bit and 8-bit environments.

## 2.1.2 Command String Initiator

A command string initiator is a delimited string of characters used in a data stream as a logical entity for control purposes. It consists of an opening delimiter (CSI), a command string (data), and a closing delimiter that is specific to the command being sent.

CSI is an 8-bit control character that can also be expressed as ESC [ when coding for a 7-bit environment.

## 2.1.3 Device Command Strings

A device command string is a delimited string of characters used in a data stream as a logical entity for control purposes. It consists of an opening delimiter (DCS), a command string (data) and a closing delimiter (ST).

DCS is an 8-bit control character that can also be expressed as ESC P when coding a 7-bit environment.

ST is an 8-bit control character that can also be expressed as ESC \ when coding a 7-bit environment.

## 2.2 Definitions

ANSI Command Structure examples:

```
ESC F
ESC <space> F
CSI Pi; Pk : ... F
DCS ... ST
```

Where:

1. ESC is ASCII character "escape", code 1BH.
2. CSI is ANSI symbol for code 9BH or the sequence 1BH 5BH.
3. DCS is ANSI symbol for code 90H or the sequence 1BH 50H.
4. ST is ANSI symbol for code 9CH or the sequence 1BH 5CH.
5. Pi, Pk are command parameters and represent ASCII encoded decimal numbers.
6. Semi-colons are used to terminate positive parameters.
7. F is a final character, ASCII codes 40H through 7FH.

**NOTE:** Spaces are used in command descriptions only for illustration, they are not part of the command. If an ASCII space character is part of a command, it will be shown as "<space>".

Sometimes it is difficult to distinguish a numeral one (1) and a lower case L (l) in the command. Check the angle of the top line for differentiation; the L has a straight line and the one has a diagonal line.

**NOTE:** Before using any print commands, check printer status (see "Reports"). Print characters are spaced with the space (SP) character. After the last printable character in a line, a carriage return & linefeed (CR, LF) or vertical tab (VT) or form feed (FF) is sent. Printable characters do not include spaces, unless they have video attributes selected.

## 2.3 Error Codes

It is very easy to make a mistake when inputting commands into a text editor, and it is often very difficult to find these mistakes later through visual inspection. The C9 now provides an error reporting system to assist users in detecting errors in received commands. If the user sends the *Command Error Reporting Enable* command, `CSI>34h`, the terminal will return an error message to the host each time an unrecognized command, improper parameter, or command with a syntax error is sent to the terminal. A full description of Command Error Reports is given in the Error Reporting Appendix.

To see a demonstration of the error reporting function;

1. Launch the communications utility (HOST.EXE) on the *Utilities and Demos* disk, then press F3, (or cycle power) to reset the terminal.
2. The terminal will reinitialize and transmit an XON (transmit on, ready to receive data) to the host. The XON appears as a left pointing triangle in the Host program.
3. Press F2 to download the ERROR.TXT to the terminal.
4. Type in (Path)\ERROR.TXT
5. Press Enter

The terminal will output the following strings (error codes) to the host.

```
ER4_27_4_t
ER6_29_1_~
```

Let's take a closer look at the ERROR.TXT file and see how these error codes are developed.

This is the program (ERROR.TXT) sent to the terminal.

```
Esc[>34h
Esc[>1;1;1;6t
EscP~1/host:local:labelEsc\
```

The first command (`Esc[>34h`) enables the error reporting function.

The second line of ERROR.TXT (`Esc[>1;1;1;6t`) is an improper text button area command. The text button area command is missing the second X coordinate. The terminal returned the error message, `ER4_27_4_t`, to the host. You can verify this error message as the first text string sent to the host on your computer display. The first portion of this error code (ER4) indicates an *Invalid Argument Count in Command* error. This makes sense considering one of the arguments (X coordinate) is missing. The second portion of the error code (27) indicates the erroneous command starts with a CSI>. The third portion of the error code (4) indicates the erroneous number of arguments. The last portion of this error code (t) indicates the string terminator in the erroneous command.

The next error message sent, **ER6\_29\_1\_~**, is a *Requested Button Number Does Not Exist On Appropriate Button Page* error. The name of this error code is self evident. Since the *Define Button Text Area* command was incorrect, no area was defined for button number 1, hence when the Define Button Response command was received by the terminal, the terminal was unable to find a matching area ID number for the button response ID number. The first portion of this error code (ER6) indicates the *Requested Button Number Does Not Exist On Appropriate Button Page* error. The second portion of this error code (29) indicates the command structure that prompted the error code. In this case it is **DCS~...Esc\**. The next portion of the error code (1) indicates the button number. The tilde (~) represents the command structure.

The user can look at CORRECT.TXT to see the proper command structure for the original program. (ERROR.TXT)

We recommend enabling the error reporting function whenever developing a new application. There are occasions when the application has an error that is not obvious from looking at the results on the terminal. The error reporting mode is an extremely helpful tool that will reduce developer trouble shooting time.



### 3.0 Getting Started With C9 Quick Mode Terminal

In this product family, C9 refers to the hardware, a serial interface printed circuit board, and Quick Mode is the name of the firmware running on the C9 board. Quick Mode firmware always runs on C9 hardware.

The C9 board also supports an earlier firmware version, called "C4 emulation" (named after an earlier hardware & firmware platform). A separate documentation is available for the C4 emulation.

Customers developing new products will want to use the Quick Mode products. C4 emulation is still available to support existing customers.

Deeco Serial Terminals - Firmware Options		
	C4 Terminal Firmware	Quickmode Terminal Firmware
<b>Serial Text Touch Terminals</b>		
ST2200 Series	yes	no
M320ST	yes	no
<b>Serial Graphics and Text Touch Terminals</b>		
M3-15	yes	no
M4ST	yes	no
M9ST Series	yes	no
M6ST	yes	no
ST3200 Series	yes	no
ST3500 Series	yes	no
ST4500	yes	yes
EM-T031	yes	yes

This Application Note was tested with an EM-T031 terminal module (Firmware version 2.05) connected to a PC compatible computer running Windows 95 and Deeco's HOST communications software, version 1.77. Even if your C9 Quick Mode unit is different, the information contained here can save you a lot of time.

This document rapidly and concisely covers:

- Power On and Self-Test
- PC Serial Communication Software
- Cabling and Connecting to a PC
- Sending Commands to the Terminal
- Sending Files to the Terminal

Finally, this document introduces Quick Assist, a PC compatible touch screen development program. Quick Assist is definitely the preferred method of developing touch applications.

### 3.1 Power-On and Self-Test

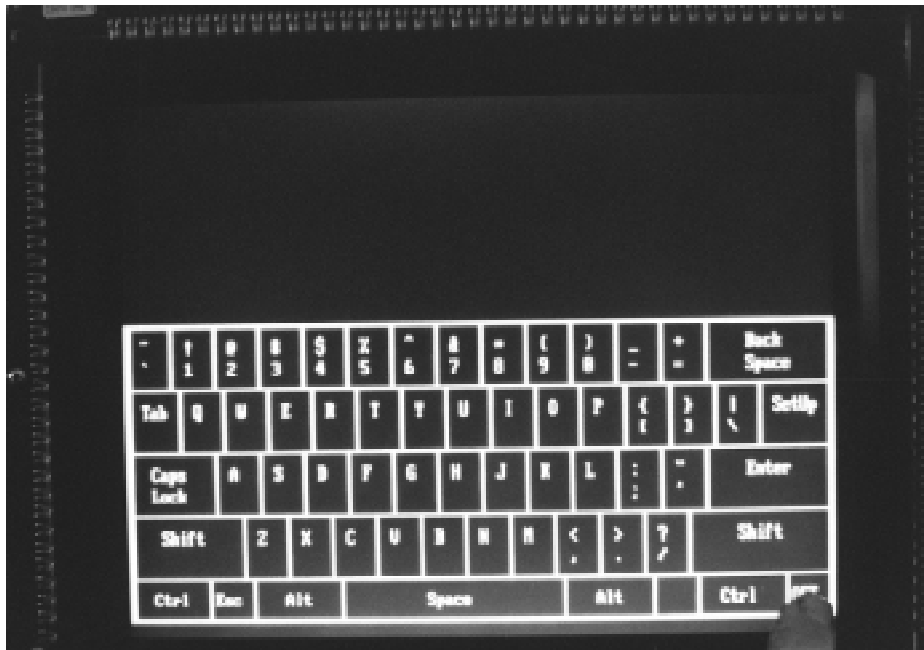
Apply power to the touch terminal as described in the User Manual for your SealTouch terminal. In most cases this consists of plugging an AC power cord into an electrical socket. Embedded touch modules ordered without power supplies require DC power.

After about 15 seconds the screen displays the message,

Quick C9 Firmware Version 2.0X  
running Self Test... OK  
Configuration by state memory

"Configuration by state memory" means that the terminal is configured by stored values. These are not necessarily factory default values - so soon we will show how to set the terminal to the factory default setting. An alternative configuration option is by executing a file (or "list") stored in non-volatile memory on the terminal. Also, after the terminal is running, and communications are established, configuration commands can be sent over the serial interface to the terminal.

***The screen saver option blanks the display after about 3 minutes of inactivity. A touch to any point on the screen restores the display.***



*Access the QWERTY Keyboard By Touching The Lower Right Corner of the Touch Screen  
Note that the SetUp key is Three keys Above the Finger, in this Photo*

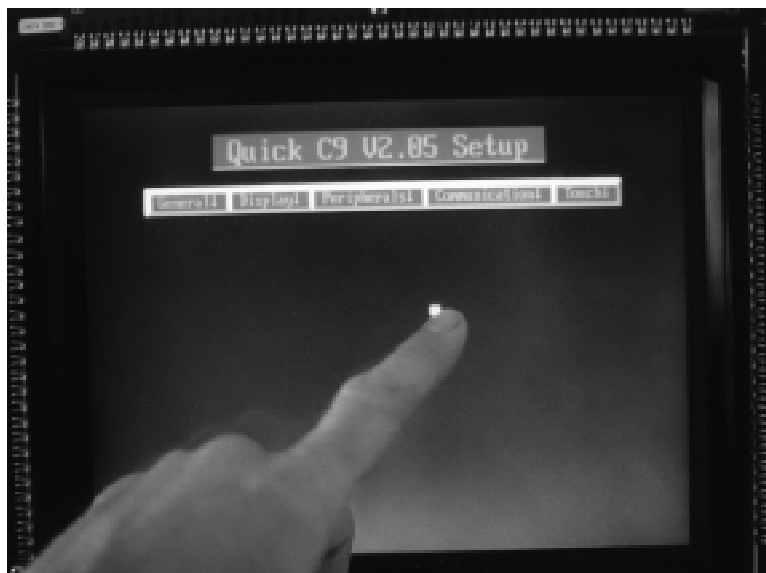
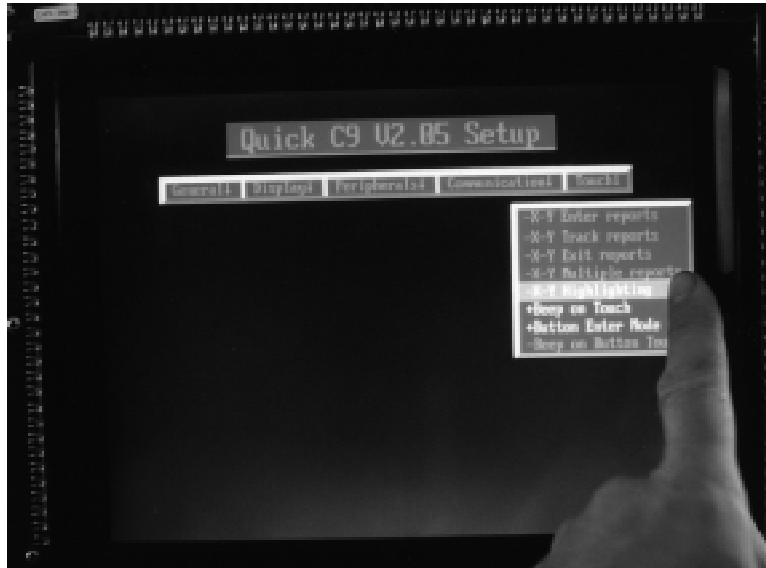
Touch the lower right corner of the touch screen. A standard QWERTY keyboard appears in the lower half of the screen, as shown above.

Touching any key highlights that key, the terminal beeps, and sends that character out the serial port. Characters do not appear on the screen. Touching the SetUp key brings up another screen, as shown in the next photo.

### 3.2 Testing the Touch System

This is a great test of the terminal, useful when first receiving the touch system, or at incoming inspection for production shipments.

From Setup, touch the "Touch" menu, then without lifting your finger, slide down the menu items until "X-Y Highlighting" is selected by the brown color bar. When you lift your finger, observe that the + or - in front of "X-Y Highlighting" changes.

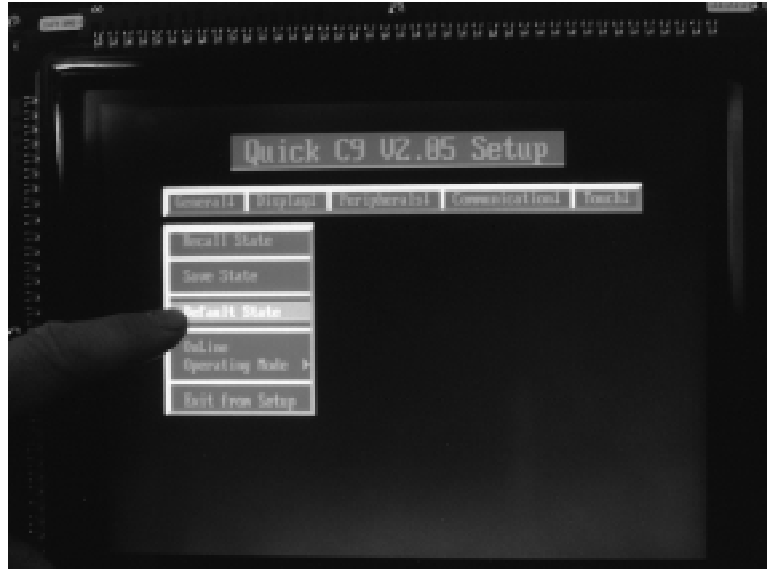


*Touch the Screen at any Point, and a Rectangle Will Indicate the Touch Location*

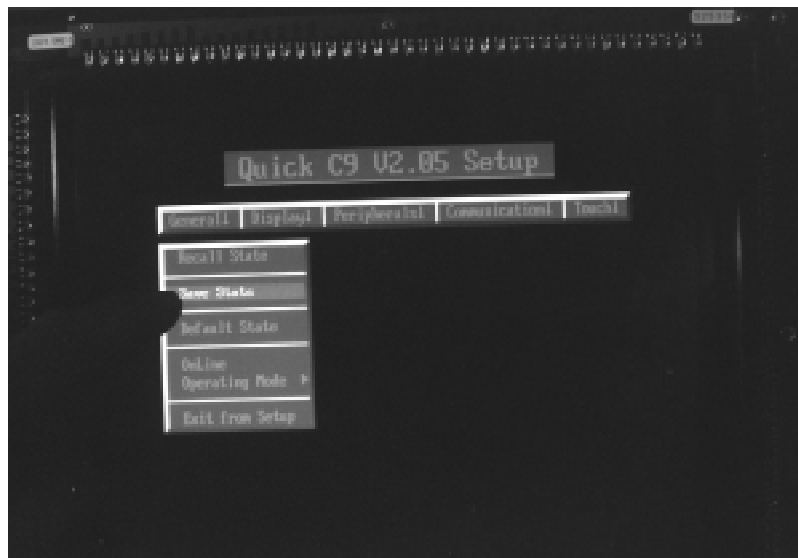
Accessing the pull-down menus in Setup is easiest if you touch the header (for example "General"), then, without lifting the finger from the screen, drag your finger through the menu, highlighting the various menu items. When you reach the command you are interested in, remove your finger from the touch screen, and the command will be executed.

### 3.3 Setting the Terminal to Factory Default Configuration

It is a good idea to be sure the terminal is in the factory default configuration. To access the stored factory configuration, touch the "General" heading, then slide your finger down to "Default State", and lift your finger. There is no feedback from the terminal, but the factory default configuration has been restored. To save this configuration through power cycles, press the "Save State" selection in the "General" menu.

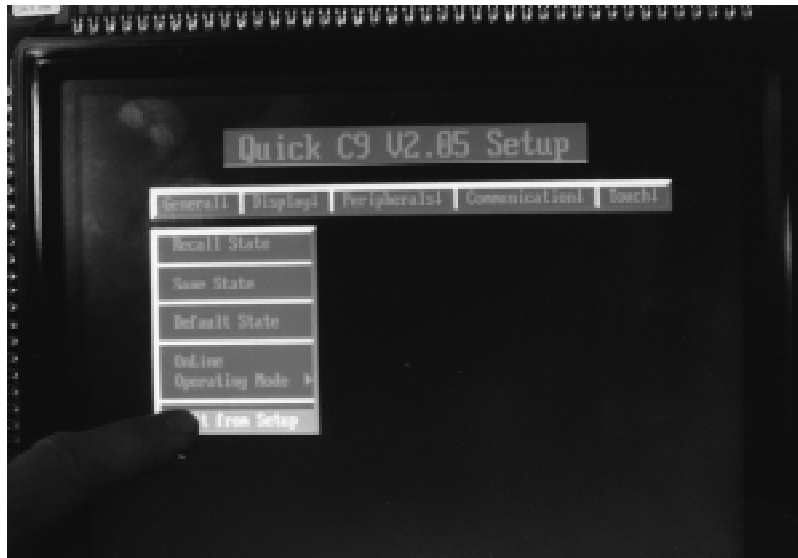


*Using The Setup Screen To Restore The Terminal To The Factory Default State*



*Activate the "Save State" Menu Selection so the Terminal Will Remain In Factory Default State through Power Cycles*

The terminal requires about 10 or 15 seconds to execute the "Save State" command, during which time the screen will not respond to touches. The easiest way to tell when "Save State" is done is to hold your finger elsewhere on the menu, and wait for the item to highlight.



*To Exit Setup Touch the "General" header, Highlight "Exit from Setup", then Lift your Finger.*

### **3.4 PC Serial Communications Software Selection**

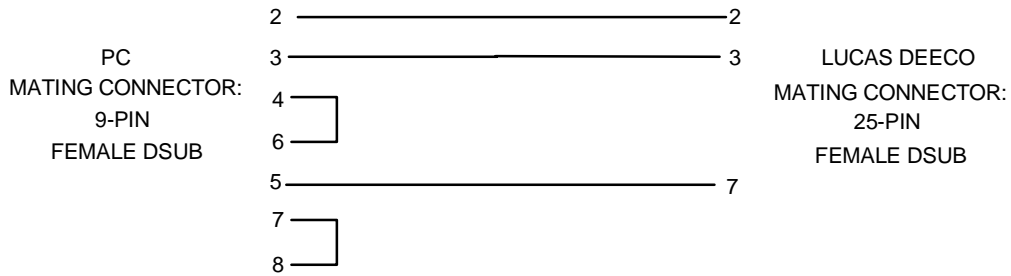
Terminal programs such as PROCOMM® and the Windows™ Terminal Accessory can be used to communicate between a PC computer and the touch terminal. However, these programs do not display all the control characters that will be sent to the terminal. Obtain HOST, a time saving PC serial communications software package from Deeco. Later we will use one of HOST's features to send files to the touch terminal.

HOST is included on the floppy disk titled "Quick C9 Utilities & Demos Disk" that is shipped with the C9 terminal. HOST is also available at no charge on the Deeco web site at <http://www.deeco.com>.

### 3.5 PC Cabling

Connect to the terminal's 25-pin DSUB connector. The 9-pin DSUB connector on the terminal is a printer output port, not an alternate serial input connector.

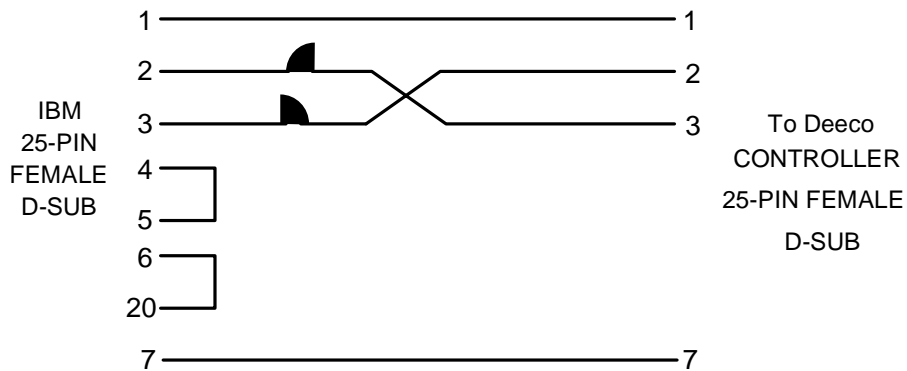
Customers require different length serial cables, constructed according to this diagram:



This is a simple RS232 connection, without hardware handshaking, and using Xon/Xoff software flow control. It is a good place to begin. The terminal also supports RS422 and RS485, and hardware handshaking, as explained in the Hardware User Manual.

Also, most commercial serial PC to PC (laptop to PC) cables will connect the 9-pin DSUB serial output of the PC to the 25-pin DSUB serial input of the SealTouch terminal.

Build this cable if you want to use the 25 pin DSUB serial connector on the PC:



### 3.6 Using A Windows 95 Computer Without A Mouse

If your PC does not have a free COM port, you can obtain a card with additional com ports. Another option is to re-boot the computer in MS-DOS mode. Here is an example of how get results if you need to navigate in Windows 95 without a mouse.

Action	Response
Disconnect your mouse from the 9 pin DSUB COM1 serial port connector, and connect the Deeco terminal	
Cycle power on the computer	Windows takes a very long time to come up
	Message appears "Windows did not detect a mouse, etc..."
Press the enter key to signal "OK"	User Name/Password may come up if Network is installed.
Press the tab key several times, and notice the different options in the window are highlighted.	
When the proper password is entered, highlight "OK" with the tab key, and press the "Enter" key	Windows continues to boot... Programs in the Start up folder are executed.
Hit the button on the keyboard with the four quadrant windows logo	Start menu and taskbar at the bottom of the display pop up
Hit the ESC key	Pop up Start menu retracts
Hit the tab key	Start button on the taskbar no longer is highlighted
Hit the "end" button on the keyboard	Last application running is highlighted
Hit the "enter" button	Now you have access to that program's window You can shut it down or whatever by using typed commands like Alt-F (if <u>F</u> ile is underlined). Then, type x (if <u>E</u> xit is a command in the pull-down menu under <u>F</u> ile) to exit the program.

Action	Response
After accessing the programs on the task bar with the "end" button you can use the tab key to gain access to any of the tasks running on the computer	
Hit the button on the keyboard with the four quadrant windows logo	Start menu pops up
Press P	Program menu opens
Press cursor to highlight MS-DOS prompt	Highlight MS-DOS icon
Press Enter key	DOS screen is activated; C:\WINDOWS
A: (or, change directories to where ever HOST is)	A:\
HOST	Host begins to run
Type any key on the PC keyboard	Character appears on the Deeco terminal
Press any key on the terminal's QWERTY touch keyboard.	Character appears on the HOST screen.
Press F7 (Ping) on the PC keyboard Host will send ESC [5n	The terminal will reply with ESC [ 0n, if communication is working properly

### 3.7 PC Configuration & Communication

Most PCs have a serial connector with male pins. Printer ports are also DSUB connectors, but have female sockets instead of male pins. If the system has a serial mouse, the mouse cable is often plugged into the 9-pin DSUB serial connector. This connector is often designated as COM1. The PC's BIOS Setup screen (which is accessible during boot-up) shows and changes the COM assignments. Later we will show how to use Microsoft Diagnostics to check COM configurations.

Connect the serial cable to any available serial port, if no serial port is available, remove the mouse or any other device from the 9-pin DSUB connector. Re-boot the computer to free the COM port.

Connect the serial communications cable between the PC and the SealTouch terminal.

Be sure the terminal is in the factory default state. Touch the "Default State" and "Save State" menu commands in Setup, as described earlier.

Run HOST from DOS.

Note: HOST's default communications port is COM1. Follow the instructions in the HOST.TXT file (also on the Quick C9 Utilities & Demos Disk) to configure Host for an alternate COM port. This can be accomplished by configuring HOST once, as well as with a command line each time HOST is run.

After HOST loads, any character typed on the PC will appear on the SealTouch Terminal, and any character pressed on the touch terminal's QWERTY keyboard will appear on the PC screen.

### 3.8 Sending Commands to the Touch Terminal

Sending commands to the touch terminal is easy. The touch terminal User Manual lists the many commands available. The commands begin with ESC, CSI, or DCS.

ANSI Sequence Introducers	Keyboard Characters	PC Screen Characters
ESC	ESC	←
CSI	ESC [	←[
DCS	ESC P	←P

To send a CSI, press the **ESC** key, located in the upper left corner of the keyboard, release the **ESC** key, then type the left bracket ( [ ) key. To send a DCS, press the **ESC** key, then the capital **P** key. First press the **ESC** key, release the ESC key, then press the **P** key. Don't press the **P** key while holding down the **ESC** key.

The most common mistake typing the CSI sequence introducer is to  
Press the **ESC** and then the [ key, without first releasing the **ESC** key

Don't type any spaces between characters. Commands in this Application Note, and in the User manual have spaces between them so they are easier to read, but always enter the commands without spaces.

Commands are case sensitive.

To return the SetUp parameters to the saved state, send this sequence from HOST:

## ESC ESC ESC ESC

```
HOST DRIVER Version 1.77, Written by Dan Miller
Copyright (c) 1995-1998 LCSP Deeco Systems

Any characters typed will be sent to the target machine.
Any characters received from the port will be echoed here.
Type 'host /?' at the DOS prompt, for command-line summary.

Special keys:
  F1 = Clear screen/Home cursor. (Deeco IR terminals only)
  F2 = Send text file to target.
  F3 = Send Hard Reset to target. (Deeco IR terminals only)
  F4 = Send CSI to target.
  F5 = Send DCS to target.
  F6 = Send ST to target.
  F7 = Send PING (-[5n] to target

  F9 = Stream data to target.
  F10 = EXIT from this program.

Serial Parameters: COM1: 9600,8,N,1, IRQ=default
Software handshaking is enabled
Transmit inter-character delay = 0 milliseconds
++++
```

*Sending four ESC commands From a PC, With HOST*

If the terminal is properly processing command strings this sequence resets the touch terminal and SELF-TEST is executed. It is a good idea to periodically send these commands when working from the keyboard.

If a hard reset (**ESC c**) is sent in a file, the touch terminal will send an Xoff to the host, but could miss part of the message while performing self test. Send four **ESCs** instead. Four Escapes is a more reliable method of resetting the unit because it is handled at an interrupt level and is never lost, regardless of the state of the command parser and internal processes. It will even reset the unit if it is hung in a loop internally, which **ESC c** will not.

### 3.9 Creating Command Files and Sending Files to the Terminal

Most problems sending command sequences to the touch terminal are caused by mis-typing. It is more efficient to construct and send files to the touch terminal than to key command sequences directly. Most text editors (not word processors) can be used to construct serial command sequence files. The MS-DOS Editor is one such text editor.

Type EDIT at the DOS prompt to access the MS-DOS Editor.

The MS-DOS Editor sees the **ESC** character as a control character, and so will not ordinarily record a command sequence. Holding the **Ctrl** key while pressing the **P** key (without releasing the **Ctrl** key) alerts the editor that the next key stroke is to be recorded. This MS-DOS Editor sequence is not case sensitive.

This sequence will create and send a file to the touch terminal:

ACTION	TYPE	RESPONSE
Change to the directory where HOST is located.	<b>CDHOST</b>	C:\HOST
At the DOS prompt, type EDIT	<b>EDIT</b>	MS-DOS Editor comes up
Alert editor to record the next keystroke. Hold the <b>Ctrl</b> key and simultaneously press the <b>p</b> key.	<b>Ctrl p</b>	none
	Press <b>ESC</b> key	←
	Press <b>[</b> key	←[
	Press <b>7</b>	←[7
Command sequence for reverse video entered. Use lower case m.	<b>m</b>	←[7m
Save as	<b>alt f</b>	menu pulls down
	<b>a</b>	Save As Menu
Enter the file name	<b>REVERSE.TXT</b>	
	Press the <b>Enter</b> key	REVERSE becomes name of file
Pull down file menu	<b>alt f</b>	
Exit MS-DOS Editor	<b>x</b>	C:\HOST
Start HOST from the DOS Prompt	<b>HOST</b>	
Press the F2 function key	<b>F2</b>	"Enter ANSI text filename..."
Enter name of file to send	<b>REVERSE.TXT</b>	
	Press the <b>Enter</b> key	sending REVERSE.TXT
Type characters on the PC keyboard		Characters typed within HOST now appear as reverse video on the terminal

**ESC [ 27m** will change to normal video, or sending four **ESC**s will reset the touch terminal to the initial state and restore normal video.

### 3.10 Sending More Files to the Terminal

There are a number of .TXT files on the "Quick C9 Utilities & Demos Disk" which show various properties of the touch terminal. The commands can be viewed and edited using the MS-DOS editor as described above, and, the files can be sent to the terminal using HOST. Also, they can be sent directly to the terminal from DOS by using the .BAT commands on the Demos Disk. Sending an **ESC c** from HOST, or cycling the power on the terminal if using the DOS .BAT commands will assure predictable results.

The Quick C9 Graphic Controller User Manual, Chapter 4 contains a good tutorial and demonstration of simple but powerful touch buttons. These touch buttons reside within the firmware of the touch terminals. Accessing this local intelligence saves programming time and transmission time.

Some files described in Chapter 4 of the Quick C9 User Manual begin with the command **ESC [0v** (set video mode to text).

To view the operation of the Scrolling Window discussed in the chapter 3 tutorial, type characters within HOST. All the typed characters will appear within a small window. When the window is full, the oldest characters will scroll off the top of the window, so there is room for the newest characters.

### 3.11 More About the "Quick C9 Utilities & Demos Disk"

The root directory contains HOST, and a number of .TXT demonstration files. The FONTOOLS subdirectory contains tools for generating custom fonts or characters. The SPRESENT subdirectory contains a demonstration of virtually all the features of the C9 Quick Mode touch terminal. The README.TXT file gives instructions on downloading the demo. The commands for the demo are contained in the SP.DL file.

This demonstration program requires 128 k of auxiliary memory on the terminal to run. You can check how much memory your touch terminal has by issuing the *System Status Request* command; `CSI < 3 n`. The response is in the form `CSI < 4; Pv; Pr; Pm; Pi; Ps; Pa n`. When Pa is 40 there is no auxiliary memory installed. Pa = 41, there is 8K; Pa=42, there is 32 K; 43 = 64 K; 44 = 128 K.

### 3.12 Quick Assist

Obtain Quick Assist software from Deeco for the easiest full scale application development. Quick Assist runs on an IBM PC, and includes a bitmap tool (located on the toolbar) which will send .BMP graphic files to the touch terminal. So, user interface graphics can be created in paint and/or draw programs. Quick Assist will place touch buttons on the screen, and includes a menu for defining the response of the buttons.

Deeco has developed graphical user interface screens for publicity photos and other purposes. Consult Applications Engineering at Deeco if you would like copies of these files for modification or inclusion in your application.

### 3.13 Practical Hints

Most problems are caused by mis-typing command strings. If a command string is unrecognized, the SealTouch terminal normally ignores it, and executes the next valid command string. C9 firmware includes a *Command Error Reporting Enable* command, `CSI > 34 h`. After this command is sent, the terminal will return an error code to the terminal. This is a great tool for application development and troubleshooting.

## 4.0 Buttons

There are a variety of objects that can be created in the Quick C9 module, to allow the user to interface with the module and with the host system. The simplest is the traditional button, a rectangular region on the screen. This is the standard interface object which has been available on all past Deeco terminal products. Another is the cascading menu, a list of one-line items which can be individually selected, and can be linked to other submenus. The third is the keyboard, which can either act like a matrix of buttons, or can be programmed to act as a buffered input device in which the user can edit the entered data before accepting it when it is correct.

These tools, or objects, have several similarities to each other; they all have defined labels, and each element can have host and/or local responses defined which are executed when the element is activated. However, each type of tools has advantages and drawbacks in certain situations.

The *button* is most useful when a small number of options are to be presented to the user, especially if the options are not directly related to each other. One major advantage of a button is that it can be made as large as desired, and can be drawn with any of the attributes (size, italics, etc.) which are available for graphics string draw. Its main drawbacks are that relationships between buttons are not clear, and large numbers of buttons on one screen give a cluttered, disorganized appearance; thus, they are not good for organizing large numbers of options.

A special type of *button* referred to as a *window* is available for displaying scrolling text onscreen within a restricted region. Scrolling buttons are usually used for displaying periodic data, rather than as user input devices.

The *menu* provides a mechanism for organizing very large numbers of choices in a very dense space. One line is devoted to each choice, and choices can be grouped together in small, related sets (as cascading submenus). They don't support larger text though, and the small lines can be both difficult to read at a distance, and tricky to accurately select.

The *buffered-style keyboard* is the only input tool which enables the user to edit the input data before accepting it. When the user must have control over his input, this is the tool of choice. However, the individual keys do not have host or local responses; the input data is handled as either host or local response, depending upon the attributes assigned to the keyboard. Although *keyboards*, like *menus*, can only display standard sized text with default attributes, the keys can be made to occupy multiple lines, which menus cannot.

The *response-style keyboard* acts literally like an array of buttons. This is a useful method for grouping related buttons together in a small space.

### 4.1 Utilities Demos And Batch Files

The utility (called Host) included on your Utilities and Demos disk is a versatile serial communications program. Read HOST.TXT on the utilities disk for full operating instructions. It provides the user with a tool to communicate with the terminal and assist in downloading various text files on the disk. The HOST.EXE program accepts command line downloading of all text (.txt) files on the Utilities and Demos disk. For the purpose of this manual the user should cycle the power on the terminal before downloading any text file or running any batch file unless otherwise instructed to do so. These files are intended as stand alone applications for the terminal. Consecutive downloading of the text or batch file without resetting the terminal or cycling power will create unpredictable results. These unpredictable results will prevent the user from reaching the objective of this section, to learn and understand the functionality of the Deeco terminal.

## 4.2 Buttons And Windows

A button may function either as a display window, providing input and feedback to the user, or as an interface device responding to a user's touch. By sending a series of commands to the screen the user can change the behavior and appearance of a button.

## 4.3 Button Structure

At a minimum, the user must send two commands to create a button on the screen. The first command establishes the area (location) of the button on the screen, the second command defines the response (output) of the button. An optional third command can be used to change the attributes of any given button, as we will see later.

To save time we recommend the user run the batch file "FIRST.BAT" located on the Utilities and Demos disk included with your shipment of the terminal. This program will transmit button commands to the terminal creating a simple button with a red label background and a blue border. After running the batch file, we recommend that you touch the button and observe the response then continue reading this section on buttons.

**NOTE:** The user should establish proper communications from their host to the terminal before running FIRST.BAT. (For instructions on how to establish communications with the terminal please refer to the [Quick Start Guide to Touch Buttons](#) in the sections entitled PC Serial Communication Software Selection, PC Cabling, and PC Configuration and Communication.

Let's take a close look at the commands needed to create a button. This example was taken from the BUTTON1.TXT file supplied on the Utilities and Demos disk. (The batch file FIRST.BAT downloads the BUTTON1.TXT program.)

```
Esc[>1;10;10;20;20t
Esc[>1;0;5;4F
Esc[>1;0;1;1F
EscP~1/host:local:label:logo Esc\
```

The first line in this example is *Define Text Button Area*:

```
Esc[>1;10;10;20;20t
```

The first portion of the command, `ESC[`, is the Control Sequence Introducer (CSI). The CSI alerts the terminal that important information is soon to follow. The first parameter (1) is the number assigned to the text button area. The next two parameters taken together (10;10) specify the upper left corner of the button area. The last two parameters taken together (20;20) specify the lower right corner of the button. The last letter (t) is a string terminator signaling the end of the command. (Please see *Define Text Button Area* in the Command Reference Appendix of this manual for further details.)

The second and third lines:

```
Esc[>1;0;5;4F
Esc[>1;0;1;1F
```

define button attributes.

All buttons, whether created as interfacing objects or display windows, are configurable, through the use of the *Define Button Attributes* command, to meet the customer's needs or tastes. The first portion of the command is the CSI once again, alerting the terminal that important information is soon to follow. The first parameter, known as the button ID number, (1) links the attribute string to the given button. The second parameter (0) corresponds to the state of the button selected. (The user, if they desire, may

design a button with several layers, each layer coming to the surface, (in rotation), with each successive button touch. MLTISTTE.BAT demonstrates this feature and you can download the file later at your leisure for a visual demonstration. For now, the user should always set this parameter to its default (0) until they are ready to experiment with multi-state buttons.) The third parameter is the attribute the user wishes to change. The user can control a variety of items on the button, such as label text color, label character size, border background and foreground color, etc. For a complete list please refer to the Command Reference Appendix. This attribute, set to the number 5, establishes control of the border background color. The next parameter (4) actually sets the background border color to red. The last character in this command (F) is the string terminator informing the terminal that this is an attribute command and signals the end of the command.

The last line defines the button responses:

```
EscP~1/host:local:label:logo Esc\
```

The first portion of the command (**ESC P**) is the Data Command Sequence. The DCS alerts the terminal that important information is soon to follow. The first parameter, known as the button ID number (1) is the assigned button number. Notice that the number is the same as the Text Button Area number. This number links the button response to the button area on the screen. This means that when the user touches within the area defined by the *Define Text Button Area* command, the button will process this button response data. We will now look at these parameters. The last parameter is a text string, "host", which defines the *host response*. Any response placed in this position of the *Define Button Response* command will be transmitted to the host via the serial port on the back of the terminal. The next parameter is another text string, "local", which defines the *local response*. Any response placed in this position of the *Define Button Response* command will be interpreted locally at the terminal, just as if the user sent the command from the host. (In our example, the terminal acts just as if you had sent the text string, "local" to the terminal from the host.) The last parameter is a text string, "label", which defines the *label* of the button. Any text placed in this position of the *Define Button Response* command will display itself as a label on the button. The last parameter is the button logo, which is displayed on the top border of the button in text mode.

We recommend the user copy BUTTON1.TXT and save the file under a different name, then experiment with the controlling parameters of the button size, responses and attributes.

Proper button structure consists of two to four commands in the following order. (Four commands are necessary for bitmap labels in graphics mode.)

1. *Define Button Area* command
2. *Define Button Attribute* command (Optional)
3. *Define Button Bitmap Label* command (Optional)
4. *Define Button Response* command

The user may send these commands out of order to the terminal and experience reasonable results, however, improper sequencing may cause problems. The *Define Button Area* command must come first in all cases. This command is the foundation upon which the rest of the button is built. If a button response, bitmap label or attribute is sent before the corresponding button area command, the terminal will simply disregard the out of order command. Note that bitmap labels can only be displayed in Graphics Mode. (If error reporting is enabled, the terminal will return the error message ER1\_1.) If the user sends an attribute or bitmap label (graphics mode only) command after the button area and response command, the terminal will redraw the button on the screen using the new updated information. This redrawing causes the button to appear to flicker momentarily on the screen. Sending the commands in the proper sequence will ensure the button is only drawn once, and the application avoids the button flickering condition. The user can witness this redrawing by downloading the batch file UNORDER.BAT. The batch file INORDER.BAT sends the exact same set of commands used in the UNORDER.BAT batch file, but sends them in the proper sequence.

Once the user has a firm understanding of button positioning and attribute control, they are ready to proceed on to creating windows.

## 4.4 Scrolling Windows

A *scrolling window* is a special type of button which acts as a small terminal window on the screen. Text which is written to a window will wrap at the right edge of the window, and all data in that window region will scroll when text is entered at the end of the screen. A button can be turned into a scrolling window by enabling the *scrolling* attribute for that button via the *Define Button Attribute* command. Scrolling windows are available in all video modes, and operate the same in each case.

There are two ways to write text to a window. The first is to select a window as the active window, using the *Set Active Button* command. Once a window is selected as active, all normal text which is received (from host port, button local responses, etc.) will be written in that window at the current cursor position.

Note that each button page has its own active window, which need not be the same as that of any other button page. By default, button 0 (a full-screen, scrolling window which is shared among ALL button pages) is the active window on each button page, but a different active window can be selected for any page, and it will be restored each time that button page is selected.

The second way to write to a scrolling window is by using the *Write Text String* command. This command can be used to update different windows without having to change the active window. For example, the following example shows how to create a scrolling window that is used as a status window, using a special status color, without regard to which window is currently active:

```
; This command creates button 5 as a scrolling window:
CSI > 5; 0; 55; 7; 79 t
; make button scrolling:
CSI > 5; 0; 13; 1 F
; turn off "touchable" attribute:
CSI > 5; 0; 14; 0 F
; no host or local response are defined because this button is NOT
; touchable. Also, no label is defined, because scrolling buttons
; don't use a label!! However, a button logo is defined in order
; to label the window with its purpose.
DCS ~ 5 /:::status window ST

; Now, with the windows created and valid, Write Text String can
; be used to write strings to the window:
DCS 25 G 5; 14; 6 AI am working GREAT!!^M ST
```

This file is located on the *Utilities & Demos* disk as STAT\_WIN.TXT.

We will now examine, one command at a time, the process of creating and activating a *scrolling window*.

To save time we recommend the user run the batch file "WINDOW1.BAT" located on the *Utilities and Demos* disk included with your shipment of the terminal. This batch file will produce a scrolling window on the terminal.

Running the batch file, WINDOW1.BAT, produces a window (button) on the screen. The user can now send information to the terminal using their host computer as the input device. The user need only type on their host keyboard and observe the appearance of their keystrokes in the newly created window.

As mentioned earlier, a button requires a minimum of two commands in order to exist. The first command allocates an area on the screen as the button area. (Note: This area may or may not react to

a touch, depending on how the user defines the button.) The user assigns a number to this touch zone as part of the `Button Area` command. Shown here is an example *Define Button Area* command.

This example was taken from the "WINDOW1.TXT" file supplied on the Utilities and Demos disk. (The batch file WINDOW1.BAT downloads the WINDOWS1.TXT program.)

```
Esc[>1;40;10;20;40B
Esc[>1;0;13;1F
Esc[>1;0;14;0F
EScP~1/| |scroll|scrollEsc\
Esc[>1b
```

**NOTE:** The first two commands for creating an active button or window are the same. If you understand these concepts, we recommend proceeding to the attribute explanation below. (That is the third line of the program.) If you are unsure of the concepts, a quick review is in order.

The first command defines button area:

```
Esc[>1;40;10;20;40B
```

The first portion of the command, (`ESC[`), is the Control Sequence Introducer. The CSI alerts the terminal that important information is soon to follow. The first parameter (1) is the number assigned to the button area. The next two parameters taken together (40;10) specify the upper left corner of the button area. The last two parameters taken together (20;40) specify the lower right corner of the button. The last letter (B) is a string terminator signaling the end of the command. (Please see *Define Button Area* in the Command Reference Appendix of this manual for further details.)

The second and third commands set button attributes, just like the previous example. In this case, they make the button *scrolling* and *not touchable*.

Now let's take a look at the fourth command in this program:

```
Esc[>1;0;14;0F
```

The fourth line of this example once again controls attributes of the window (button). The last two parameters (14) and (0) control the touchable mode of the window. If the touchable parameter is set to 0, as it is in this case, the button will not respond to a touch. This is known as the touchable attribute on the Deeco terminal.

The fourth line defines responses for the window:

```
EScP~1/| |scroll|scrollEsc\
```

The first portion of the command, (`ESC P`), is the Data Command Sequence (DCS), as before. The first parameter is the assigned button number, which must be the same as the button number provided to *Define Button Area*. This number links the button response to the button area on the screen. The *host* and *local* responses are not actually used in this case because this button is not touchable, and thus will not return responses. The separator character is still required to delimit the various responses. Also, the *label* is not used with scrolling windows, though it's a good idea to provide one anyway in case the screen gets switched to graphics mode, where no scrolling buttons exist. The *logo* is especially important for scrolling windows because no label will be displayed to identify the window.

The last line selects the active button:

```
Esc[>1b
```

In this case it chooses button number 1 as the active button. By choosing button number 1 as the active button, any text sent to the terminal will display on button (window) number 1.

**AN IMPORTANT NOTE ABOUT ACTIVE BUTTONS:** Button ID number 0 is reserved on the C9 Quick Mode terminal. The user cannot create a button with this ID number. The terminal actually creates a non touchable, active button on the screen at startup. This button encompasses the entire screen. This functionality allows the terminal to display text when no user buttons (windows) are set as active buttons.

## 4.5 Logos

The Deeco Quick C9 terminal provides the user with a new button feature called a logo. The logo is an additional label which is displayed centered on the top of the button. This feature provides additional information about the button's purpose. It is especially useful in labeling scrolling windows, where the normal button label is not used. Run the batch file LOGOS.BAT and take a quick look at the logos created.

This is the content of the LOGOS.TXT file.

```
Esc[0v
Esc[>1;10;10;12;19t
Esc[>2;10;20;12;29t
Esc[>3;10;30;12;39t
Esc[>4;10;40;12;49t
EscP~1/Host:Local:ON:Motor1Esc\
EscP~2/Host:Local:OFF:Motor1Esc\
EscP~3/Host:Local:ON:Motor2Esc\
EscP~4/Host:Local:OFF:Motor2Esc\
```

Line 1 is the *Set Video Mode* command enabling text mode on the terminal:

```
Esc[0v
```

The next four lines define button areas, using *Define Text Button Area*:

```
Esc[>1;10;10;12;19t
Esc[>2;10;20;12;29t
Esc[>3;10;30;12;39t
Esc[>4;10;40;12;49t
```

(For a discussion of the *Define Text Button Area* command, refer to the Command Reference section).

The second portion of the program consists of the *Define Button Response* commands.

Look at the first *Define Button Response* command:

```
EscP~1/Host:Local:ON:Motor1Esc\
```

Notice this response is created in exactly the same manners as described earlier in the Button Structure section. The *Define Button Response* command contains the *DCS* (*EscP*), the *button number* (1), the *host response*(Host), the *local response*(Local), the *label*(Label), and the logo (Motor1), followed by the string terminator.



the number seven (7) for the next four pixel locations. The number 7 translates to white on the 16-color palette. The last 28 pixel locations are ones (1s) again. This is the very bottom of the "chin" on our alien friend. The next line consists of 27 ones (1s) followed by 6 sevens (7s), followed by 27 ones (1s) again. This line is the fourth line from the bottom of the picture adding to the structure of the alien being. You can examine the image data in detail by looking at the file BITMAP.TXT with a text editor.

The last line once again defines button responses:

```
EscP~1/Alien:I come in Peace^M:GreyEsc\
```

Note the control code (^M) in the local response. This sends a carriage return (CR) locally to the terminal after the ASCII (text) characters.

As you can see this is a somewhat tedious process. The customer can certainly create custom bitmaps for their programs in a text file, but we highly recommend the use of the Deeco development tool Quick Assist. Please contact your local sales rep to obtain this tool.

## 4.7 Window (Button) Layering And Updating

The new Deeco terminal gives the user the ability to layer windows (buttons). (This function is only available in Text mode.) This means the customer may stack one window on top of another or allow portions of windows to overlap with no loss in screen information. This feature is especially useful for providing help menus, hidden messages, and important updates. To see an example of layered windows run the batch file LAYERED.BAT. Touch each window to raise it to the top of the other windows. Text can be sent to any scrolling window, even if it is partially or wholly obscured.

This allows the host to display status messages, help screens, and other transient information on the screen, without worrying about preserving data which is behind it.

There are two ways to write to a given window, the *Set Active Button* command or the *Write Text String* command. The *Set Active Button* command will tell the terminal where to direct incoming text. Note the user must enable the scrolling attribute on any given button before it will accept incoming text from the host. Run the LAYERED2.BAT batch file to see how the *Raise Button To Top* and *Set Active Button* commands can interact with each other.

Here is a listing of the first example:

```
Esc[0v
Esc[>1;1;10;8;25t
Esc[>2;1;15;8;30t
Esc[>3;6;10;14;25t
Esc[>4;6;15;14;30t
Esc[>1;0;13;1F
Esc[>2;0;13;1F
Esc[>3;0;13;1F
Esc[>4;0;13;1F
EscP~1/Host1::This is^MButton One:button1Esc\
EscP~2/Host2::This is^MButton Two:button2Esc\
EscP~3/Host3::This is^MButton Three:button3Esc\
EscP~4/Host4::This is^MButton Four:button4Esc\
Esc[>1b
```

Line 1 is the *Set Video Mode* command enabling text mode on the terminal:

```
Esc[0v
```

The next few commands (*Define Text Button Area* commands) define the text button areas for each window just as they did in our earlier examples.

```
Esc[>1;1;10;8;25t
Esc[>2;1;15;8;30t
Esc[>3;6;10;14;25t
Esc[>4;6;15;14;30t
```

The first command following the *Define Button Text Area* commands is the *Set Button Attribute* command.

```
Esc[>1;0;13;1F
```

Notice that this attribute command tells the terminal to enable scrolling for button (window) number 1. You must enable scrolling in order to send text to the window (button).

The rest of the program sets the attributes for the balance of the windows and defines the button response for each window.

```
Esc[>2;0;13;1F
Esc[>3;0;13;1F
Esc[>4;0;13;1F
EscP~1/Host1::This is^MButton One:button1Esc\
EscP~2/Host2::This is^MButton Two:button2Esc\
EscP~3/Host3::This is^MButton Three:button3Esc\
EscP~4/Host4::This is^MButton Four:button4Esc\
Esc[>1b
```

The last line in the program sets button number 1 as the active button.

```
Esc[>1b
```

This is not an absolutely necessary command, but it is highly recommended the user establish a habit of setting the desired active button at the end of any program containing scrolling windows. The terminal defaults to button zero as the active window if no scrolling windows are created, otherwise the last scrolling window created becomes the active window.

The second command for sending text to a scrolling window, *Write Text String* (see Command Reference section), is a shortcut command. It allows the user to write text to any scrolling window, without changing the active window. The user can also control the foreground and background text colors through the *Write Text String* command. This command is very useful for quick updates, status reports and alerting the end user to alarm conditions. We recommend the user run the LAYERED.BAT batch file once again, but this time, rather than using the buttons to control the windows, experiment with the *Write Text String* command. The batch file RESTRNT.BAT demonstrates the usefulness of the *Write Text String* command as well.

## 4.8 Button Pages

### Current Page

Buttons and windows as stand alone items are very useful, but grouping buttons together in a coordinated effort, will help create a more powerful application to the end user. The Deeco terminal provides this grouping ability in the form of button pages. Please take a moment to run the PAGES.BAT batch file and observe how the use of button pages can guide and assist the user. There are two distinct reference pages available in the terminal. These pages are called the current page and the definition page. The current (button) page is always the page displayed on the screen. The definition page is the page that is affected by commands sent either from the host or locally from a button push. For instance, let's assume we have a program that contains 3 button pages. On the first page there are two buttons, one labeled "Lions" and one labeled "Lambs". This first page is the current (displayed) page. Depending upon the button pushed, you will call up either page two or page three. If the user pushes the Lion button, page two then appears on the screen with information about lions and a return button that takes the user back to the original page. If the user pushes the Lamb button, page three then appears on the screen with information about Lambs and a return button that takes the user back to the original page. Let's examine the first choice, when the user pushes the Lion button. The Lion button response, (see BUTTONS section for details on button responses), contains an embedded command that says make page two the current page. The terminal will now display page two on the screen. If the user pushes the Lambs button, the exact same process take place, only the Lamb button is preprogrammed to make page three the current page. Buttons labeled RETURN are present on both pages two and three. These RETURN buttons are programmed identically to make page one the current page. Please run the LIONS.BAT batch file and keep this current page concept in mind as you push the buttons.

### Definition Page

The definition page is the page affected by button commands or the commands sent by the host to the terminal. If the user needed to change a button characteristic or response in the background, away from the current (displayed) page, he / she would make this change on the definition page. For instance, referring to our earlier Lion and Lamb example, if the user wanted to update the text on the Lion page, (page two), while still displaying page one, the user could send the command `CSI>1;2P`. This command says make, (or keep), page one as the current page, and make page two the definition page. This functionality provides the user a system for updating buttons in the background, undetected by the end user while they are making choices on the current page. Please run LIONS.BAT again and push either button on page one. Now return to page one, then while page one is displayed on the terminal screen, run the LIONUP.BAT batch file. This file uses the *Set Button Pages* commands `CSI>1;2P` and `CSI>1;3P` to update the Lion and Lamb buttons (windows). Now push either button on page one to observe the changes. One last note on current and definition pages; the current and definition page can be one and the same if the user so desires. The user can send the command `CSI>1;1P` which tells the terminal to make page one the current page and also make page one the definition page. (The user may also use a shortcut command `CSI>1P` to produce the same results. Sending `CSI>1P` is the same as sending `CSI>1;1P`). If the user makes a given page both the current and definition page, it just means that the changes made will be visible to the user at the time of the change.

## 4.9 Multi-State Buttons

Multi-state buttons are buttons designed to change their response after each touch. A multi-state button gives the user the ability to logically layer button responses (or states) on top of one another. Each touch of the button produces the appropriate output for the current state, then brings forth the next button state. To visualize this, imagine a stack of common playing cards sitting on a table face-up. Let's say the first card, the card on top of the deck, is the ace of hearts, you then touch the top card (ace of hearts) and move it to the bottom of the deck. The next card in the deck is now exposed. Let's say it's the seven of clubs. You can now touch this card and then move it to the bottom of the stack. You can continue this process, eventually bringing the original card back to the top of the deck (stack). Multi-state buttons work in the same manner. The top button (state) responds as programmed, then moves to the bottom of the stack after the push, and this brings forth the next state. The user can define different responses for each state. This layering technique allows the user to set up a program that will walk someone through a given routine in a specific order. Run MLTISTTE.BAT to see a small demonstration of a multi-state button.

Let's take a look at the structure of the multi-state button command.

```
Esc[1v
Esc[>1;20;20;50;50B
EscP~1/host1:local1:Ace^Mof^MHearts
/host2:local2:Seven^Mof^MClubs/host3:local3:Six^Mof^MDiamonds
/host4:local4:Nine^Mof^MSpadesEsc\
```

This example shows one button with four distinct states. The button area is allocated in exactly the same manner as any other text button, using the *Define Button Area* command. It is the changes to the *Define Button Response* command that actually create the multi-state button. The first portion of the command is the DCS announcing the beginning of a command to the terminal. The second portion of the command (1) is the button number linking the response to the button area (touch zone). The next portion of the command (**host1:local1:Ace^Mof^MHearts**) is the button *Host* and *Local* response, and *Label* for the first state of the button (state zero). (Note the use of the control code **^M** to add a carriage return to the label producing two lines of text.) Notice, this command structure (so far) is identical to the structure used to create a normal button. The next portion of the command (**host2:local2:Seven^Mof^MClubs**), separated by a forward slash, is the button *Host* and *Local* response and *Label* for the second state of the button (state one). A forward slash separates each portion which consists of the button *Host and Local* response and *Label* for each state defined. In effect, all the user needs to do to create a multi-state button, is to add additional responses onto the end of an existing *Define Button Response* command string and separate these new responses with a forward slash.

There is one more area of interest concerning the multi-state button: the area of attributes. The user may assign unique attributes for each state of the multi-state button. To accomplish this task simply use the *Define Button Attributes (Single Form)* command and select the state you wish to customize. The batch file STATTRIB.BAT demonstrates this capability, and the example listed below shows the user where to make the appropriate changes.

```
Esc[1v
Esc[>1;20;20;50;50B
Esc[>1;0;4;4F
Esc[>1;0;10;2F
Esc[>1;1;4;0F
Esc[>1;1;10;2F
Esc[>1;2;4;4F
Esc[>1;2;10;2F
Esc[>1;3;4;0F
Esc[>1;3;10;2F
EscP~1/host1:local1: Ace^Mof^MHearts
/host2:local2:Seven^Mof^MClubs/host3:local3:Six^Mof^MDiamonds
/host4:local4:Nine^Mof^MSpadesEsc\
```

For a complete breakdown of possible button attributes, refer to the Command Reference section.

## 4.10 Multi-State Button Example Walk Through

Here's one example of how a multi-state button defined with the proper attributes might prove useful. Let's say the Deeco terminal is in use as the man machine interface, (MMI) to a programmable logic controller, (PLC), controlling a stationary crane. This crane is manually controlled by a resident operator. The crane picks up scrap metal from the beds of trucks. The crane then travels on a fixed bar to deliver the scrap to a melting pot. It's important that the crane release button is unavailable to the operator until the crane is in position over the drop-off point. A simple multi-state button could help control the capture and release of the crane. Turning off the touchable attribute on the multi-state button, when the button is in its release state, (release function is on top of the stack), would prevent the operator from releasing the scrap load early. The multi-state button would remain untouchable until the host received confirmation that the crane was in the proper position. Once the host received this confirmation, the host could then send an attribute command to turn on, (set), the touchable mode and highlight the multi-state button in release mode, alerting the operator that the crane was now in proper position for release. (Note: The user could design this functionality into separate buttons, avoiding the use of multi-state buttons, however the multi-state button along with controlling the order of pushes, (responses), saves space on the terminal screen, maximizes system memory usage and allows the user to group related functions on one button.) For a quick demonstration of this functionality, run the CAPTURE.BAT batch file. After running the CAPTURE.BAT file, push the capture button, notice how it cycles to the release state. Now push the button again, several times if you like, notice the button does not respond. Now, without turning off the terminal, run the RELEASE.BAT batch file. Notice the button highlights, alerting the user, and will now respond to your touch. After responding to the touch, the multi-state button resets (turns off) the touchable mode once again. The button will not respond in the release state until the host signals it to do so. (In our case running the RELEASE.BAT batch file.)

This is a small example of the terminal's capabilities when the user combines the multi-state button with other features. In this example we combined multi-state buttons attribute commands and the ability to redefine buttons on the fly. These features were demonstrated in earlier examples, but become much more powerful when used together.

## 4.11 Limitations And Restrictions Of Multi-State Buttons

The system limits multi-state buttons to 256 separate states for any given button. Attributes 13-17 are not assignable to each given state; they apply to every state of the multi-state button.

## 4.12 Button Zero (0)

Button Zero (0) is reserved for system usage. Button zero is always present on the screen. By default it is the entire size of the screen, created without a border or label and with the X-Y Coordinate Report mode disabled. The user may enable the X-Y Reporting to send X-Y coordinates out the serial port. With this mode enabled, the terminal will send X-Y coordinates to the host whenever the user touches a free area of the screen. Free area is defined as any portion of the screen that does not contain a user defined button. Button zero is needed for layering buttons and windows on the screen.

## 4.13 Text And Graphics Video Mode

The Deeco terminal functions in two distinct video modes, text and graphics. The default mode is graphics. With windowing, text scrolling, button highlighting and user defined attributes, text mode allows for a very flexible and fast application environment. Screen draws, page changes, and button responses are faster in text mode. If aesthetics are the user's primary concern, we recommend using graphics mode. Graphics mode offers the user a more aesthetically pleasing environment and the ability to control button border widths. For a complete list of button capabilities and restrictions in either given mode please see the Command Reference and the section entitled Limitations, Restrictions and Characteristics.

In text mode, the button borders and highlights are drawn using text ASCII characters. Button areas, in text mode, are referenced against text cell positions on the display. (25 X 80 text cell positions) In graphics mode, the button area is referenced against available touch zones on the display. (59 X 80) If the user creates a button on the terminal, then switches back and forth from Text mode to Graphics mode, they will see a slight positional change of the button(s). Running the batch file, POSITION.BAT, then alternatively sending the set video mode commands (`Esc[0v` and `Esc[1v`) from the HOST utility will demonstrate this effect. The terminal provides two button area commands to the user. The first command is Define Text Button Area, the second is *Define Button Area* (see Command Reference section). The *Define Text Button Area* command is provided to improve button alignment in text mode. If the user's application uses both text and graphics mode, then the user should create buttons using the *Define Text Button Area* command to prevent buttons from overlapping. The *Define Button Area* command is provided for all graphic applications and any time the user needs to increase the total number of possible button areas on the screen. Since the graphics mode has a finer resolution for button area coordinates in the Y direction, the user can put a greater number of total buttons from top to bottom in graphics mode than in text mode. To prevent accidental overlapping of buttons, we recommend the user always create buttons using the *Define Text Button Area* command except for applications that absolutely need the greater number of buttons areas per page capacity.

## 4.14 Copy Button Command

There are many button commands available to the user in the Deeco terminal, but one special command is of particular interest. This command is the *Copy Button Response* command (see Command Reference section). This command lets the user copy the responses of a button on one page to a button on another page. In essence the user can duplicate a button or buttons on several pages. When a user copies a button, as opposed to creating a new button, the terminal links the copied button to the original button's attributes and responses. The terminal uses less memory by not having to re-create duplicate attribute and response lists. Copying buttons becomes extremely memory efficient when using the same button on several pages. Run the batch file COPYRSPN.BAT to demonstrate the copy command.

Let's take a closer look at the COPYRSPN.TXT file.

```
Esc[>1;10;10;20;20;B
Esc[>2;10;21;20;31;B
Esc[>3;10;32;20;42;B
Esc[>4;10;43;20;53;B
Esc[>0P
EscP~1/host:^[[>1P:goto^Mpage 1Esc\
EscP~2/host:^[[>2P:goto^Mpage 2Esc\
EscP~3/host:^[[>3P:goto^Mpage 3Esc\
Esc[>1P
Esc[>2;2;0C
Esc[>3;3;0C
EscP~4/host:^[[>0P:return^Mto mainEsc\
Esc[>2P
Esc[>1;1;0C
Esc[>3;3;0C
Esc[>4;4;1C
Esc[>3P
Esc[>1;1;0C
Esc[>2;2;0C
Esc[>4;4;1C
Esc[>0P
```

The first four lines in this program allocate the button area for each button in the program.

```
Esc[>1;10;10;20;20;B
Esc[>2;10;21;20;31;B
Esc[>3;10;32;20;42;B
Esc[>4;10;43;20;53;B
```

The next lines sets page zero (0) as the current (display) and definition page. This command instructs the terminal to display page 0 and define new buttons on page 0.

```
Esc[>0P
```

The next set of lines are the *Define Button Attribute* and *Define Button Response* commands for page zero.

```
EscP~1/host:^[[>1P:goto^Mpage 1Esc\
EscP~2/host:^[[>2P:goto^Mpage 2Esc\
EscP~3/host:^[[>3P:goto^Mpage 3Esc\
```

The next lines sets page two as the current (display) and definition page. This command instructs the terminal to display page one and define the buttons on page one.

```
Esc[>1P
```

The next set of commands in our example are *Copy Button Response* commands.

```
Esc[>2;2;0C
Esc[>3;3;0C
```

The first command (**Esc[>2;2;0C**), copies the response from button number two (2) created on page zero (0) to button number two (2) on page one. The next command (**Esc[>3;3;0C**), copies the response from button number three (3) created on page zero (0) to button number three (3) on page one. The *Copy Button Response* for our first example, (**Esc[>2;2;0C**), breaks down as follows. The first portion of the command (CSI) is the Control Sequence Introducer, alerting the terminal that important information is soon to follow. The second portion (2) is the button number for the new button the user is creating. The third portion (2) is the old or source button. The source button is the response the user wishes to copy, (it is the original button). The fourth portion of the command (0) is the button page for the source (original) button. The last portion of the command (C) is the string terminator signaling the end of the command to the terminal.

If you are in the mood to experiment, try running the following batch files (COPYRSP2.BAT and NORMAL.BAT), one at a time, then send the *Get Button Free Space* command (**CSI>5n**) after each batch file to see the difference in system ram usage

#### **4.15 Erase Screen / Clear Active Button**

Since the Deeco terminal display consists of a full screen button (button zero) as the default configuration, the erase screen button command (**CSI2J**) actually erases the full-screen window (button zero). If the user wishes to clear the active window (button) in their application they need to use the *Clear Active Button* command (**CSI<e**).

#### **4.16 Limitations Restrictions And Characteristics**

The user may define up to 250 separate button areas (touch zones). These button areas are global and available to the user on all pages of the application. It is unrealistic to define this many buttons on a single page, to do so would mean that each button created would be too small to use, however 250 buttons per page is the system numerical restriction in affect.

The user may define up to 250 separate button pages total in any given application. System ram limitations will usually override the numerical limitations imposed by the software. The number of buttons and the size of each individual button response determines the total size of the customer's program. This total size is restricted by the amount of physical ram installed in the unit.

The system limits multi-state buttons to 256 separate states for any given button. The user cannot set attributes 13-17 for individual states on multi-state buttons.



## 5.0 Menus, Organization And Attributes

A *Menu* is a collection of *Menu Items*. Each *Menu Item* is a discrete execution unit which functions much like a *Button*; it has a host response, local response, and label options associated with it, all of which function exactly like the button items of the same names. Additionally, each *Menu Item* has an optional Next Menu value associated with it, which specifies a submenu that will be displayed automatically when a given menu item is highlighted; this creates cascading menus similar to those found in Microsoft Windows 95.

Both the menus and the individual menu items have attributes associated with them, which include colors as well as menu orientation, border style, etc. Menu item attributes include colors, separator and inactive types. These are documented fully in the Command Reference.

### Next Menu identifier

The *Define Menu Responses* command has an optional argument for each menu item, called the Next Menu number. This number, running from 0-250, specifies an additional menu which will be drawn when the related menu item is highlighted. Menu items which indicate a Next Menu value will NOT process their host/local responses when selected; they only cascade to the next Menu.

## 5.1 Menu Creation Sequence

A menu should be created and drawn with the following sequence of commands:

*Define Menu Responses*  
*Define Menu Attributes (optional)*  
*Draw Menu*

Note the differences between button definition and menu definition: for menus, attributes (if any) are set AFTER responses are defined (not before). There is NO *Define Menu Location* command; this optional data is included in the *Define Menu Responses* command. Also, the menu is NOT drawn automatically when *Define Menu Responses* is received; an explicit *Draw Menu* command is required.

There are, conceptually, two different kinds of menus in this system; a Top Menu is drawn by sending a *Draw Menu* command after the appropriate menu has been created. A Sub Menu is drawn automatically when a menu item is highlighted which has a *Next Menu* value defined. There is no actual difference in creating the two menus, and in fact any given menu could be used as a Top or Sub menu; the difference is only one of concept; usually a menu tree will have one top-level menu that is intended to be initially drawn, and all other menus in it will normally only be seen when the parent menu is highlighted.

The (Y;X) arguments to the *Define Menu Responses* command are optional, and are not used when drawing a submenu; the position of the submenu is derived from the position of the parent menu item. If a menu is intended to be a submenu, a few bytes can be saved in the command string by omitting these coordinates.

### Example Menu program

The following example shows a simple, two-level menu system. Note that each of these commands is documented more fully later in the Command Reference. As in most Deeco command examples, spaces in the commands are for reading clarity only, and should not be typed into the actual file. This file is located in the *Utilities and Demos* disk as **MENU0.TXT**.

```
; Define Top Menu, with three items.
; Note that the definition for item two contains %23,
; which means "Menu 23 cascades from this menu item".
; There are three initial parameters for this menu, specifying
; menu number (1) and upper-left corner position of menu (10;30)
; IN TEXT COORDINATES.
DCS% 1;10;30 /i1|l1|line1 /i2|l2|line2%23 /i3|l3|line3 ST

; Define the Sub Menu, also with three items.
; Because this menu is intended to be a Sub Menu, no corner
; position is specified; its position will be derived from the
; position of the menu item which instantiates it.
DCS% 23 /s1|s1|submenu1 /s2|s2|submenu2 /s3|s3|submenu3 ST

; Draw Top Menu
CSI >3;1M
```

As a further clarification, it should be noted that the menu system has NO relationship with the button-page system. Selecting and changing button pages will NOT change or undraw a drawn menu.

## 6.0 Keyboards

Two types of keyboards are available on the Quick C9. The Static keyboards are built into the module and cannot be modified by the user. The User-Defined keyboards, as the name suggests, are defined by a series of commands from the host.

### 6.1 Static Keyboards

There are three static keyboards on the C9:

- The QWERTY keyboard (keyboard number 0) is laid out like a standard typewriter keyboard. It is a *response* style keyboard and has special keys to switch to the KEYPAD keyboard.
- The NUMPAD keyboard (keyboard number 1) is no longer supported on the Quick C9. If a keyboard with this functionality is required, it can be created using user-defined keyboards, documented elsewhere. However, keyboard number 1 is still reserved by the Quick C9, and cannot be assigned by the user as a user-defined keyboard.
- The NUMERIC keyboard (keyboard number 2) is a *buffered* style keyboard which collects numeric inputs and returns them to the caller once input is accepted. This keyboard is used by certain SETUP functions, and can also be called by the host.

The static keyboards can be drawn using the Draw Keyboard command, as well as by using the hotkey area on the screen. However, they cannot be modified by any of the keyboard definition commands.

### 6.2 Keyboard Styles

User-defined keyboards offer an alternative to buttons and menus for presenting options to users. A keyboard can be set up to operate in one of two styles, called *response style* and *buffered style*.

#### Response-style keyboards

In response style, *host* and *local* responses are defined for each key by the user, and are processed when a key is pressed. This operation is identical to the manner in which buttons and menu items work, and when defined this way, the keyboard acts exactly like a matrix of buttons. The following example file which is listed on the *Utilities and Demos* disk as KBD\_RESP, demonstrates the commands required to create and draw a response-style keyboard:

```
; define a response-style keyboard. This keyboard has two
; rows of three keys each, with implied key-width specifiers.
DCS ? 3;10;10
&k1:k1:k1 /k2:k2:k2
&k3:k3:k3 /k4:k4:k4
ST

; set attributes for keyboard and keys
CSI > 3;-1;0;11;1 K
CSI > 3;0;3;1;1;3 K
CSI > 3;1;3;1;1;3 K
CSI > 3;2;3;1;1;3 K
CSI > 3;3;3;1;1;3 K
CSI > 3;3 K
```

## Buffered-style keyboards

In buffered style, inputs from each key press are displayed in a buffer at the top of the keyboard. The user can backspace to correct inputs, and accepts the input when it is correct. Once the user accepts the buffered data, it is processed as either host or local response, depending on the attributes of the keyboard.

The first row of a *buffered* keyboard **must** contain only ONE key, with a blank label (use one or more spaces as label). This row becomes the display line for the buffered input.

Only the first character of the host response for each key is used as the input for that key, with the exception of control keys, which consist of two characters. There are two control keys defined for buffered style keyboards:  $\wedge M$  indicates *accept input*, and  $\wedge H$  indicates *delete last character entered*. The remainder of the host response (and all of the local response, if any) is ignored.

The example file KBD\_BUFF on the *Utilities and Demos* disk demonstrates the various commands required to create a buffered keyboard.

There is a special attribute for buffered keyboards, called *password* mode. When *password* attribute is enabled for a keyboard, any characters which are entered by the user will be displayed as star characters (\*), rather than the actual characters which the user entered. The characters can still be erased and retyped by the user (if a delete key has been placed on the keyboard), and the actual characters entered will be processed normally once the user accepts the data.

### 6.3 Creating And Modifying A Keyboard

Three commands may be used to create and draw a keyboard. These commands are:

*Define Keyboard Responses*  
*Define Keyboard Attributes*  
*Draw Keyboard*

*Define Keyboard Responses* defines the number of rows and keys in the keyboard, and the responses of each key. *Define Keyboard Attributes* is used to specify colors, keyboard style, and other characteristics of the keyboard and keys. The *Draw Keyboard* command is used, naturally, to actually draw the keyboard once it is defined as desired.

The format of the Define Keyboard Responses command is:

```
DCS ? Pid ; Py ; Px
    & responses % width
    / responses % width
    / responses % width
    & responses % width
    / responses % width
    / responses % width
ST
```

where *responses* represents `host_response | local_response | label` in the same format as responses are defined for buttons and menu items. `Pid` is the keyboard number, and `Py;Px` are the position of the upper-left corner of the keyboard **in text coordinates**. As usual with documentation for Deeco commands, the entire command string from DCS to ST represents one command; spaces and carriage returns are for clarity only.

Note that the `\&'` character means “begin a new key on a new row”, and the `\/'` character means “begin a new key on the current row”.

The `%width` specifier is optional; if it is omitted, the key width is set to `string_length+2` to allow for spaces between the label and key boundaries. The width of the entire keyboard is determined by the width of the widest row; if other rows are shorter, the LAST key on the row will be extended to force the row to the correct width. Thus, for example, to create the required blank top row on a buffered keyboard, use the string `& : :SPACE`. This key is by itself on the row, so is inherently the last key on the row; the width specifier is omitted, so the width of the key, and the row, will be derived from the width of the entire keyboard. No host or local responses are defined for this “key”, since it is not truly a key at all, but just a display buffer.

The number of lines in a key label can be increased by including `^M` in the label. For example, the label `Back^MSpace` will create a two-line key with `Back` on the first line and `Space` on the second line. The height of a keyboard row is determined by the key with the largest number of lines on that row.

## 6.4 Keyboard Hotkey

The QWERTY keyboard can be presented onscreen either by touching the hotkey corner, or by sending the Keyboard On command `CSI > 32 h` to the terminal. The hotkey corner is a 2x2-touch-zone area in the lower-right corner of the screen. The QWERTY keyboard, as well as the KEYPAD keyboard which is accessed from it, can be removed again either by touching the OFF key or sending the *Keyboard Off* command `CSI > 32 l` to the terminal.

## 6.5 Access To Keyboard And Setup Screen

Access to the QWERTY keyboard, as well as the Setup Screen, can be controlled with keyboard enable/disable commands:

<i>Keyboard Enable</i>	<code>CSI &gt; 33 h</code>	Enables the QWERTY keyboard
<i>Keyboard Disable</i>	<code>CSI &gt; 33 l</code>	Disables the QWERTY keyboard
<i>Setup Key Enable</i>	<code>CSI &lt; 18 h</code>	Enables SETUP key on QWERTY keyboard
<i>Setup Key Disable</i>	<code>CSI &lt; 18 l</code>	Disable SETUP key on QWERTY keyboard

## 6.6 Drawing Keyboards

Any keyboard, including the static keyboards, can drawn by sending the *Draw Keyboard* command. Note that disabling keyboard with the commands discussed in the previous sections does NOT prevent drawing them with *Draw Keyboard*.



## 7.0 Touch System Operation

### 7.1 Touch Modes

The terminal supports several basic types of touch reporting. These are discussed in the following paragraphs.

**Entry Mode:** sends a report of the coordinates where a finger first breaks the I-R beams. If this is the only active reporting mode, a user could touch the screen and then drag his finger around, but the only report would be the point of first contact.

**Exit Mode:** sends a report of the coordinates where a finger last breaks the I-R beams. If this is the only active reporting mode, a user could touch the screen and then drag his finger around, but the only report would be made after the finger is lifted from the screen.

**Track Mode:** sends a report of the coordinates of a finger at each change of positions as it moves around the screen. If Entry Mode is on, it will also report the location of the first touch. This mode has no fixed reporting period, but instead, reports when the finger moves.

**Multiple Touch Mode:** sends an error report whenever the user touches the screen in more than one place at a time.

**Highlight Screen Touch Mode:** draws a touch cursor on screen wherever the screen is touched. The size of each region is one touch zone. This mode is most useful when used in conjunction with the Exit Mode, as it allows the user to "fine tune" his touch position before activating it.

**Beep on Report Mode:** causes the terminal to emit an audible click each time a touch report is sent to the host.

The command format for setting these modes is:

```
CSI > Pm h      to set
CSI > Pm l      to clear
```

Multiple modes may be set or cleared with one command:

```
CSI > Pm; Pm; Pm; . . . Pm h  to set
CSI > Pm; Pm; Pm; . . . Pm l  to clear
```

Where:

**Pm** is an ASCII two-digit number:

16	Entry Mode
17	Exit Mode
18	Track Mode
19	Multiple Touch Mode
20	Highlight Touch Mode
21	Beep on Touch Report

All of the above modes, except 20 and 21, are initialized in the cleared state.

All modes will be returned to their default settings by the command:

```
CSI > 6 D
```

## 7.2 Touch Reports

Touch reports for the various modes are reported in ANSI compatible formats. All three touch modes have this report format:

**CSI > P1; Pc Fn**

Where:

**P1** is the vertical touch coordinate in decimal ASCII. range = 1 to 59

**Pc** is the horizontal touch coordinate in decimal ASCII. range = 1 to 80

**Fn** is the final letter of the command which designates the type of report:

**E** Entry Report

**X** Exit Report

**T** Track Report

The Multiple Touch error report has this report format:

**CSI > M**

Each touch position corresponds to a character on the display screen.

## 7.3 Touch Inquiry

The touch sensor normally sends touch information, depending on which modes are active, on an unsolicited basis; that is, only when the user touches the screen. It is possible for a host system to solicit the touch sensor about current finger position or latest reported finger position.

To request the current finger position send the command:

**CSI > 0 n**

The system will respond with a standard track mode report if there is something blocking the beams:

**CSI > P1; Pc T**

If there is nothing currently touching the screen it will respond:

**CSI > T**

To repeat the most recently transmitted touch report from any mode, send the command:

**CSI > 1 n**

The sensor will respond with the appropriate report type. If no touch report modes are active, it will respond as if the current finger position had been requested.

## 7.4 Bad-Beam Notification

The touch system is designed to automatically detect bad beams in the IR array. The default operation is to stop using the bad beams, but continue to test them in case they begin operating normally at a later time. No bad beam reporting is performed by default, although the host can request a report of current bad beams using the *Inquire Beam Failure* command.

In addition, the module can be programmed to notify the user of any bad beams, either by beeping or by sending a report back to the host. This programming can be done either via the setup screen, or with the following commands:

**CSI > Pt T** (*Set Bad Beam Beep Time*) Set bad-beam reporting rate (in minutes), Pt is between 0-60, with 0 disabling ALL reporting mechanisms.

“Set Beep on Bad Beam” is enabled automatically, whenever this value is greater than 0.

**CSI > 44 h/1** (*Set Bad Beam Report Enable/Disable*) Enables/Disables reporting to host, if reporting rate is > 0.



## 8.0 Graphics Commands & Techniques

The Quick C9 supports a full set of graphics drawing primitives, including:

- Cursor positioning
- Vector (line) draw, X-Y and polar, vector draw to
- Rectangle draw, rectangle draw with autofill, block draw
- Circle draw, circle draw with autofill, circular arc draw
- Pattern fill
- Graphics string draw, with various parameters

Most graphics commands have both *relative* and *absolute* forms. An *absolute* command explicitly specifies the starting position of the command. For example, *Vector Draw Absolute* command `DCS 3 G 100; 100; 200; 200 ST` specifies the starting and ending points of the line. A *relative* command does not specify a starting position; instead, it uses the current graphic cursor position (often called the CP) as the assumed starting point. For example, the *Vector Draw Relative* command `DCS 3 G 50; 50 ST` draws a line 50 pixels right and 50 pixels up from the CP.

Relative drawing commands are useful for creating *relocatable graphics objects*. Most graphics commands leave the CP at the end of their extent; for example, after executing a relative or absolute *Vector Draw* command, the CP is located at the ending point of the line. Thus, by connecting graphics primitives together with relative drawing and positioning commands, you can create a graphical object that can be drawn identically anywhere on the screen just by positioning the cursor and drawing that command sequence. This works very well if the commands to draw your graphical object are embedded in a display list.

The following example uses relative drawing and positioning commands to draw a meter on the screen with the upper-left corner located wherever the CP is placed. This file is located in the *Utilities and Demos* disk as METER.TXT. There are also two files which download this meter to a display list (METER.DL) and then render it at several different screen positions (METER.DRW).

```
; draw square on screen
DCS 6G160;130 ST          ; Rectangle Draw Relative

; draw label
DCS 36G2;3;1 ST          ; set graphics text color
DCS 2G-30;-121 ST        ; Cursor Position Relative
DCS 12GA memory usage ST ; Graphics String Draw Relative

; draw gauge face
DCS 36G 4;12;0 ST        ; set graphics draw color
DCS 2G -110;-56 ST       ; Cursor Position Relative
DCS 4G 40;100;56 ST      ; Vector Draw Polar Partial, Relative
                          ; (CP is not changed)
DCS 4G 40;100;124 ST     ; Vector Draw Polar Partial, Relative
DCS 8G 56;124;40 ST      ; Arc Draw Relative (CP not changed)
DCS 8G 56;124;100 ST     ; Arc Draw Relative

; pattern-fill region
DCS 2G 50;0 ST           ; Cursor Position Relative
DCS 36G 4;2;0 ST         ; set graphics draw color
DCS 19G 1 ST             ; set fill pattern
DCS 11G ST               ; Pattern Fill Relative
DCS 19G 0 ST             ; reset fill pattern

; install nut to hold meter needle
DCS 2G -50;0 ST          ; Cursor Position Relative
DCS 36G 4;4;0 ST         ; set graphics draw color
DCS 7G 5;0 ST            ; Circle Draw Relative with Autofill

; add meter needle
DCS 36G 4;13;0 ST        ; Cursor Position Relative
DCS 4G 6;100;121 ST      ; Vector Draw Polar Partial, Relative
```

## 8.1 User-Definable Line Style And Fill Pattern

The Quick C9 allows the user to specify their own line style and fill pattern. The line style is used by the Vector Draw, Rectangle Draw, Circle Draw and Arc Draw routines to render their lines. The fill pattern is used by Pattern Fill, Block Draw, Circle Draw with Autofill, and Rectangle Draw with Autofill to draw their interior regions. However, creating the data for these commands requires some knowledge of binary and hexadecimal data. This section will give examples of creating a user-defined line style and fill pattern. The Quick Assist program, available from Deeco, provides an alternate method for developing these commands.

### User-defined line style

The line style is a series of 16 bits that are repeatedly drawn end-to-end to create a line. To create a user-defined line style, specify the pattern of 16 bits for the pattern, then convert the 16-bit pattern into a decimal number which will be sent to the C9 as part of the command. Here is an example:

1. Select a line pattern

The easy way to do this is draw a pattern on graph paper so it looks familiar, then replace marked bits with 1s and cleared bits with 0s. The pattern for this example is:



1 0 1 1 0 0 1 1 1 0 0 0 1 1 1 1 ← == in binary

2. Convert to decimal

101100110001111 binary = B38F hex = 45967 decimal

3. Generate *Set User-defined Line Style* command

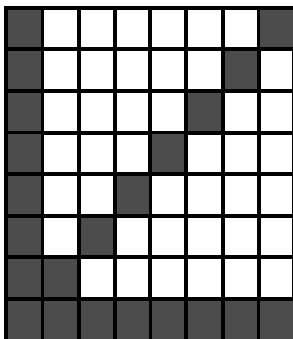
`DCS 13 G 45967 ST`

### User-defined fill pattern

The fill pattern is generated in a similar fashion, except it is a sequence of eight 8-bit patterns, drawn from top to bottom. In other words, the fill pattern is an 8x8 block of bits. Here is an example:

1. Select a fill pattern

The easy way to do this is draw a pattern on graph paper so it looks familiar, then replace marked bits with 1s and cleared bits with 0s. The pattern for this example is:



- Convert to binary, then to decimal

row	binary	decimal
0	1000001	129
1	1000010	130
2	1000100	132
3	1001000	136
4	10010000	144
5	10100000	160
6	11000000	192
7	11111111	255

- Generate *Set User-defined Fill Pattern* command  
`DCS 13 G 129; 130; 132; 136; 144; 160; 192; 255 ST`

The file LINEFILL.TXT on the Utilities and Demos disk defines both of these patterns, and demonstrate drawing an object with them.

## 8.2 Graphics Clipping

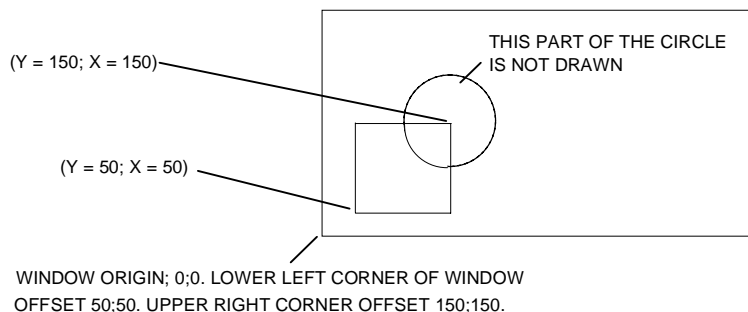
To “clip” an object is to draw only that portion of the object that lies within the “graphic window”. Portions of the object lying outside the window are not drawn, and are therefore “clipped” at the window’s boundaries.

Clipping is automatically disabled by default, and may be enabled via the *Graphics Clipping Enable* command.

Figure 8-1 demonstrates clipping. The small rectangle represents the boundaries of the “Graphic Window”. The origin of the Graphic Window is set as being equal to the Global Origin (0;0 in global coordinates).

The lower left corner of the graphic window is offset from the graphic window origin by Y=50; X=50. The upper right corner of the Graphic window is offset from the graphic window origin by Y=150; X=150. This creates a Graphic window measuring 100 pixels by 100 pixels.

A circle is drawn with a large enough radius that part of it lies outside the boundaries of the “window”. When clipping is enabled, the part of the circle lying outside the window will not be drawn in raster memory.



**Figure 8.1 Graphic Window Operation**

### 8.3 Erasing Graphics

All graphics, text and graphic, are drawn in raster memory by changing the contents of raster memory. The new data to be drawn typically replaces the old data already in the raster, thus overwriting whatever was there. This is accomplished by using the *Set Graphics Write Mode* command.

There are 3 alternative parameters to the standard replacement operation, known as “complement”, “set”, and “clear” write modes. The write mode specifies a logical operation performed between the old raster data and the new raster data.

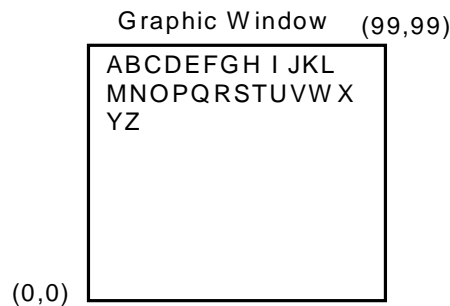
In complement write mode, the new and old data are exclusively “ORed” together. The result then replaces the old data. In set write mode, the old and new data are inclusively “ORed” together. Clear write mode combines the two data by “ANDing” the old data with the logical complement of the new data.

The complement mode is particularly powerful in that it allows selective erasure of objects with the raster. For instance, if a vector (solid line) exists in raster memory, it can be erased by selecting complement mode, and then re-drawing the exact vector. The result is the erasure of the vector.

### 8.4 Graphic Text

Text may be drawn within the graphic window as well as such objects as vectors, circles, and rectangles. Characters drawn using a graphic command are different from those drawn using the terminal mode. Graphic text may be started on any pixel boundary, whereas alpha text is restricted to the character rows and columns of the terminal. Graphic text may be rotated on any 45 degree angle. Also, graphic text may be confined, by clipping or auto-wrap, to the graphic window.

Restricting graphic text to the graphic window allows the window to operate like a “mini-terminal”. It is not possible to scroll the window, but it still can operate much like a small terminal. To illustrate, take the example of a 100 x 100 graphic window and a text string consisting of the uppercase alphabet. With auto-wrap on, and the cursor beginning in the upper left corner of the window, the resulting text string would appear as shown in figure 8-2.



**Figure 8-2 Graphical Text Auto-Wrap**

The standard character cell width is eight pixels; the window is 100 pixels wide, so the window can accommodate 100/8, or 12, characters. The 13<sup>th</sup> would be clipped. Using auto-wrap, the 13<sup>th</sup> through 24<sup>th</sup> characters are placed on the next character line within the window.

A character line is one standard cell height, or 19 pixels tall. Since the window is 100 pixels tall, 5 character lines would fit within it. The graphic window then behaves like a 5 row, 12 column terminal (without scroll). There is even a cursor much like the alpha cursor which can be displayed. Cursor movement commands like Line Feed and Carriage Return move the cursor within the window.

## 8.5 Fast Vector Mode

The C9 can be directed to enter a mode where its sole responsibility is to draw a connected set of vectors. The host must supply the vector end-point information. A vector is drawn from the previous end-point to the new end-point. This mode is known as the “Fast Vector Mode”.

The controller will acknowledge entering the Fast Vector mode by sending the ASCII control code “ACK” (code 06H). The host must wait for this response before sending any vector information.

To draw a vector, the host must specify a Y, X coordinate. The controller will respond by drawing a vector from the last point that the host specified in this mode, to the new point. The first point sent results in a vector drawn from the window origin to that point.

In Fast Vector mode, all coordinates are treated as global coordinates.

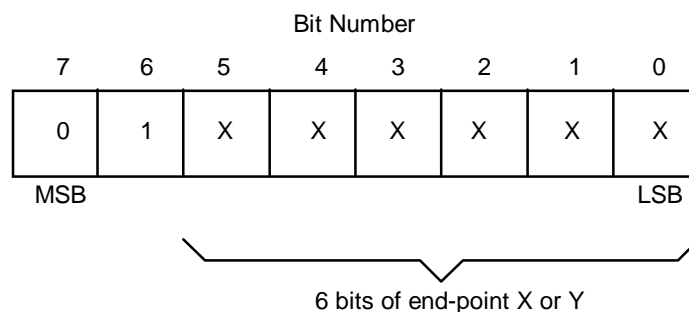
The host may specify the line style with which the vector is drawn. Included in the selection of line styles is a “black” line style. This line style is invisible against a black background so it may be used to simply move to a new point without connecting it to the old point, or for erasing an existing vector.

An end-point specification consists of one to six bytes of information. The first byte is known as the command byte. It specifies how many additional bytes will follow. The remaining bytes may or may not be sent by the host, depending on the command byte. These bytes specify two things: 1) line style, and 2) end-point. If the line style is sent, it immediately follows the command byte. It must be sent only if the line style is to change. The initial line style is solid foreground.

End-point information is encoded in four bytes. The Y and X components of the end point are 12-bit integers. The 12 bits of information are split into two 6-bit values. These values are sent to the controller in two separate bytes. The 6 bits of information are always adjusted toward the least significant bit within the byte.

Bit number 6 of the byte is always set and bit 7 is always reset. Setting bit 6 prevents ASCII control codes from being sent inadvertently to the controller. The four resulting bytes are termed High Y, Low Y, High X, Low X corresponding to which 6 bits the byte contains. The order of transmission is High Y, Low Y, High X, Low X. All bytes need not be sent. A byte must be sent only if the 6 bits of information it carries differs from the last end-point’s corresponding 6 bits.

All points must be specified in global coordinates. The initial point is taken to be the graphics cursor (in global coordinates).



**Figure 8-3 Fast Vector Bit Location**

The command byte tells the controller whether the line style byte will be sent and which of the four end-point bytes will be sent. This information is bit encoded within the command byte. A bit in the command byte is set if one of the bytes is to follow and is reset if the byte will not follow. The command byte bit structure is as follows:

Bit Number	Byte to Follow
0	Low X
1	High X
2	Low Y
3	High Y
4	Line Style
5	Reserved (0)
6	Always Set (1)
7	Always Reset (0)

**Table 8-1 Fast Vector Bit Assignments**

The line style code is encoded in the least significant 3 bits of the Line Style Byte. Bit number 6 of the Line Style Byte is always set, and bit number 7 is always reset. Bits 3 through 5 are ignored, but should be reset. The line styles selected by bits 0, 1, and 2 are as follows:

Bit Number 210	Line Style
000	Solid Foreground
001	Long Dash
010	Short Dash
011	Dotted
100	Dot-Dashed
101	Sparse Dot
110	Solid Background
111	Solid Background

**Table 8-2 Fast Vector Line Styles**

Fast vector mode is exited when the command byte specifies that no bytes will follow.

### **Example 1:**

The following example shows how to convert display coordinates into the Fast Vector format.

Convert Y = 525 to Fast Vector notation

**Step 1)** Convert decimal coordinate to binary.

525d = 001000001101

**Step 2)** Split binary value between the 6<sup>th</sup> and 7<sup>th</sup> place setting to create separate values.

001000 (upper)          001101 (lower)

**Step 3)** Convert each 6 bit value into an 8 bit value with bit 7 always reset and bit 6 always set.

01001000 (upper)      01001101 (lower)

**Step 4)** Convert binary values to hex.

01001000=48h          01001101b=4Dh

**Step 5)** Convert hex values into their ASCII equivalents.

48h=H                  4Dh=M

The values HM are the Fast Vector notation for the Y=525 coordinate.

### **Example 2:**

The following example shows how to draw a border on the C9 using Fast Vectors.

**Step 1)** Move the graphics cursor to (0,0)          **DCS 1 G 0 ; 0 ST**

**Step 2)** Enter Fast Vector Mode.                  **DCS 34 G ST**

**Step 3)** Enter fast vector coordinates.

(Coordinates should be preceded by the control byte in bold.)

**LLr**      Draws vector from graphic cursor position (0,0) to (479,0).  
Control byte L specifies a change in the Y coordinate only.

**Cl~**      Draws a vector from (479,0) to (479,639).  
Control byte C specifies a change in the X coordinate only.

**Lfc**      Draws a vector from (479,639) to (0,639).

**C@@**      Draws a vector from (0,639) to (0,0).

For reference, the control byte character to draw a vector using an X and Y component is a capital letter O.

**Step 4)** Exit Fast Vector Mode.

**@**          Specifies that no bytes will follow. Exits from Fast Vector Mode.

## **8.6 Auxiliary Memory And Display Lists**

Display lists are sequences of commands stored in auxiliary memory. These lists are stored in auxiliary memory. On power-up the controller determines the type and size of Auxiliary memory installed by reading jumper settings. The auxiliary memory size can be determined by using the *System Status Request* or *Auxiliary Memory Checksum Request* command.

A display list may be “executed” by the controller via the *Display List Execute* command. The host specifies the list to be executed by its ID number. A executable list must a display list type, not a data or font list type.

Any list type may be read by the host. The controller will, upon direction by the host, transmit the body, encoded in hex-ASCII, of a list to the host.

Each display list also contains a one-byte checksum, which is a “modulo 256” (or 8-bit) sum of all the bytes in the display list. The user can read this checksum using the *Display List Get Information* command, and use it to ensure that the data which is store agrees with what they wrote. The checksum is also verified when a *Display List Execute* command is received. If the checksum is wrong, the display list is not executed.

### 8.6.1 Other Auxiliary Memory List Types

There are two other types of data which can be stored auxiliary memory:

- Font Lists: These are user-defined characters set bitmaps which can be used to display many different characters sets on the screen simultaneously. They are discussed in the section on User-Definable Fonts.
- Bitmap Lists: These are bitmap images which can be stored in auxiliary memory and then drawn very quickly by a separate Draw Bitmap List command. Bitmap Lists are discussed later in the section on Drawing Bitmaps.

## 8.7 User-Definable Fonts

The Quick C9 provides enhanced font handling to assist European customers and other customers who must handle numerous character sets simultaneously. This improved character set handling will be provided via two new features:

- A new type of data list called a *font list* will be available, which enables a user to download a complete bitmapped character set (or any desired portion of it) into the auxiliary memory. The number of character sets which can be stored into auxiliary memory is limited by the size of the installed auxmem devices, and the range of characters which are stored for a given character set.
- There will now be four user-definable character sets available, rather than one fixed set and one user-defined set.

A command will be provided to load any font list into any one of the four character sets.

This copy operation is very quick so the user can switch freely between any of the stored font lists, with little impact on performance. The *Character Set Select* and *Character Attribute Select* commands will now accept values 0-3 for character set number. The *Define Alternate Character* command will accept an extra parameter at the beginning of the arguments, which will specify which character set to modify.

### 8.7.1 Font Lists

Users will now be able to download bitmapped font files into auxiliary memory for permanent storage, using a special kind of *data list* called a *font list*. An MSDOS program called FONTDL.EXE will be provided by Deeco to perform this font download. Any bitmapped font file of the correct size can be downloaded using FONTDL, and the user can specify the range of characters to copy from the font file (for example, one might choose to download only the characters from 80hex to 9Fhex). The number of fonts which can be stored in auxmem at one time is limited by the size of the installed auxmem device, and the number of characters which are stored from each font.

**Example:**

Each character requires 19 bytes to store. If the full range of characters from 32 to 255 is stored for each font, and 128K auxmem is installed, 30 different fonts could be stored in font lists at one time, and loaded into one of the four character sets as needed.

A font list is loaded into a character set using the *Load Font List* command (`CSI < font_list ; character_set f`). The *Load Font List* command executes very quickly (on the order of 1 millisecond for a full character set), so the user can switch between different font lists freely, with little impact on performance.

**8.7.2 Auto-loading character sets**

When the C9 is rebooted, it searches the auxiliary memory for font lists numbered 1, 2, and 3. If any of these are found, they are automatically loaded into user-defined character sets 1,2,3 respectively. Any character sets which do NOT have corresponding font lists will be initialized to the default VGA font. This allows the user to set up as many as three fonts which are ready for use immediately upon starting up.

**8.7.3 Defining User-defined characters**

The alternate character set refers to the set comprised of characters defined by the host. A character image consists of n bytes which define an 8 x n pixel array. For 640 x 480 displays, n is 19. When the character is drawn, this image is transferred to raster memory at the designated point. The character is accessed by 1) invoking the alternate character set, and 2) identifying the character by its 8-bit ASCII code.

This example will demonstrate the process of re-defining a character in the alternate set. In this example, we wish to create a character with the bitmap shown below. Since our application does not use the tilde symbol (~, ASCII code 126), we will replace this character with our bitmap. In this example, it is assumed that the display in use is 640 x 480 in size, which means the character bitmap contains 19 rows. The following table shows the character sizes for various Deeco video displays.

Display Size	Font Rows
640 x 480	19
640 x 400	16
640 x 350	14

The new character will look like this:

```

  **
  **

*****
**  **
**  *
**
**  *
*****
**  *
**
**
**  *
**  **
*****

```

The steps involved in translating a character bitmap into a *Define Alternate Character* command are:

1. Draw the character on graph paper. Remember that the characters are 8 bits wide and 19 bits tall (the height will depend on the display in use, as mentioned above). Place a 1 in each position which has a bit visible, and 0 in each position that is not visible.
2. To the right of each row, write the same binary, but BACKWARDS. The C9 font definition routines require the data to be reversed.
3. Next, write the decimal equivalent of the reversed binary data. These decimal values are the stream of data that must be passed to the controller, to redefine the character.

Here is an example of this process for the character shown above:

Character Diagram	Binary Version	Reversed Binary	Decimal Value
**	00001100	00110000	48
**	00011000	00011000	24
	00000000	00000000	0
*****	11111110	01111111	127
** **	01100110	01100110	102
** *	01100010	01000110	70
**	01100000	00000110	6
** *	01101000	00010110	22
****	01111000	00011110	30
** *	01101000	00010110	22
**	01100000	00000110	6
**	01100000	00000110	6
** *	01100010	01000110	70
** **	01100110	01100110	102
*****	11111110	01111111	127
	00000000	00000000	0
	00000000	00000000	0
	00000000	00000000	0
	00000000	00000000	0

Now, we can create the C9 command to redefine our character:

```
DCS33G1;126;48;24;0;127;102;70;6;22;30;22;6;6;70;102;127;0;0;0;0ST
```

Then, when we switch to the Alternate character set, typing '~' will display our new character instead. The following C9 command sequence will create the character and display it in the center of the screen:

```
DCS 33G1;126;48;24;0;127;102;70;6;22;30;22;6;6;70;102;127;0;0;0;0ST
```

```
CSI < 1 S <--- selects alternate character set
```

```
CSI 12;40H <--- Positions cursor in center of screen
```

```
~ <--- Type our redefined character, using alternate character set
```

## 8.8 Drawing Bitmap Images

There are three different techniques which can be used to draw a bitmap image on the Quick C9. Each has advantages and limitations. Images can be stored as 16-color or 256-color images, though the method used to determine which type of image is being sent varies with the method; see the Command Reference chapter for detailed information on the commands which are used for each method.

- **Button Bitmap Labels**  
A button can be created, then assigned a bitmap label via the *Define Button Bitmap Label* command. Whenever the button page containing that button is drawn, the button with its bitmap label will be drawn.

### Advantages:

- ⇒ The image will draw very quickly.
- ⇒ In many cases, a bitmap is a very effective way of labeling a functional object.

### Disadvantages:

- ⇒ The ONLY way to draw the image is to draw the button page; thus no other button page can be drawn while this image is present.
- ⇒ The image size is limited to 64Kbytes.
- ⇒ All the resources of a button are consumed; if the only thing that is desired is to draw an image, this is an inefficient method for accomplishing it.

- **Draw Bitmap command**  
The *Draw Bitmap* command can be used to draw an image on the screen.

### Advantages:

- ⇒ There is no limitation on the size of the bitmap (other than screen size). This is the only way to draw a full-screen bitmap image.

### Disadvantages:

- ⇒ Because the image is drawn as it is received over the serial port, this is an extremely slow way to draw an image! A full-screen, 640x480x16 color image, transferred at 9600 baud, would require over five minutes to complete.
- ⇒ Because the image is not stored internally in QC9 memory, it cannot be redrawn if it is overwritten by a button-page change or any other mechanism. It would have to be downloaded via the serial port again.

- **Bitmap Lists**  
The bitmap image is downloaded over the serial port ONCE, using the *Define Bitmap List* command, and stored in auxiliary memory. It can then be drawn at any time, using the *Draw Bitmap List* command. This is the only way to permanently store images in the Quick C9.

Advantages:

- ⇒ The image data is stored in auxiliary memory, so it is not lost when power is off.
- ⇒ The image is drawn very quickly.
- ⇒ The image is not associated with button pages, so it can be drawn at any time, regardless of current button page.
- ⇒ Image can be used in *Bitmap List File* and *Bitmap List Fill* operations (see below).

Disadvantages:

- ⇒ The image size is limited to 64Kbytes.
- ⇒ There must be sufficient available auxiliary memory for the image; this limits the number of images which can be stored in the module.

### **8.8.1 Tiling regions with Bitmap Lists**

Bitmap Lists can be used to fill a region with a repeating pattern, much like *Pattern Fill* and *Block Draw* are used. The difference here is that *Bitmap List Tile* and *Bitmap List Fill* use the specified Bitmap List to fill the region, rather than a standard fill pattern.

## 8.9 Graphic Lists

A Graphic List is a new way of storing groups of graphics commands in the C9. If a normal graphics command is sent to the unit (for example, Draw Vector Absolute), the line is drawn and then promptly forgotten; if one wishes to draw the same line again, the command would have to be repeated. This can require a significant amount of time for a complex graphic object. A Graphic List provides an alternative method for drawing such a graphic object; download the list of graphics commands to a Graphic List, then draw it using the Graphic List Draw command. Valid graphic list numbers are 0-65534. The following commands will be supported in graphic lists:

Valid Commands in Graphic Lists	
Command	Description
DCS 0 G...ST	Clear Graphics Window
DCS 1 G...ST	Set Graphic Cursor Position, Absolute
DCS 2 G...ST	Set Graphic Cursor Position, Relative
DCS 3 G...ST	Vector Draw
DCS 4 G...ST	Vector Draw Polar
DCS 5 G...ST	Vector Draw To
DCS 6 G...ST	Rectangle Draw
DCS 7 G...ST	Circle Draw
DCS 8 G...ST	Arc Draw
DCS 9 G...ST	Block Draw
DCS 11 G...ST	Pattern Fill
DCS 12 G...ST	Graphic String Draw
DCS 13 G...ST	Set User-Defined Line Style or Fill Pattern
DCS 14 G ...ST	Ellipse Draw, Relative/Absolute
DCS 15 G...ST	Draw Bitmap List
DCS 16 G...ST	Set Graphics Character Attributes
DCS 17 G...ST	Set Graphics String Attributes
DCS 18 G...ST	Set Line Style Attribute
DCS 19 G...ST	Set Fill Pattern Attribute
DCS 20 G...ST	Set Graphic Write Mode
DCS 23 G...ST	Draw Graphic Object, Absolute/Relative
DCS 24 G ...ST	Bitmap List Tile
DCS 25 G...ST	Write Text String
DCS 35 G ...ST	Bitmap List Fill, Absolute/Relative
DCS 36 G...ST	Set Color Attributes
DCS 37 G ST	Pop Graphic Cursor Position
DCS 37 G 1 ST	Push Graphic Cursor Position

As an example, we will re-use the METER.TXT example from the beginning of this chapter, store the commands as a graphic list, then draw it at different locations on the screen:

Step 1: create the graphic list (METER1.GL)

```
; open graphic list 1
DCS 1g

; draw square on screen
DCS 36G4;10;0 ST
DCS 18G0;6 ST
DCS 6G160;130 ST
DCS 18G0;1 ST
; draw label
DCS 36G2;3;1 ST
DCS 2G-30;-121 ST
DCS 12GA memory usage ST

; draw gauge face
DCS 36G4;12;0 ST
DCS 2G-110;-56 ST
DCS 4G40;100;56 ST
DCS 4G40;100;124 ST
DCS 8G56;124;40 ST
DCS 8G56;124;100 ST
; pattern-fill region
DCS 2G50;0 ST
DCS 36G4;2;0 ST
DCS 19G1 ST
DCS 11G ST

; install nut to hold meter needle
DCS 2G-50;0 ST
DCS 36G4;4;0 ST
DCS 7G5;6 ST

; add meter needle
DCS 36G4;13;0 ST
DCS 4G6;100;121 ST
; close the graphic list
DCS g
```

Step 2: draw the graphic list in several places (METER1.DRW)

```
; execute graphic list 1 at several locations
DCS 1;200;50x
DCS 1;200;200x
DCS 1;200;350x
DCS 1;200;500x
```

You can obtain a listing of all currently-defined graphic lists in the module by sending the Graphic List Report command (defined in the Command Reference), using HOST or a similar terminal program.

### **8.9.1 Linking A Graphic List To A Button Page**

Another feature is the ability to link a graphic list to a button page. Once this is done, every time that button page is drawn, the linked graphic list is automatically executed; this provides a way to handle graphics and buttons as a single related set. The file GL\_BTNS.TXT on the Utilities and Demos disk provides an example of this linking ability.

### **8.9.2 Alternatives To Graphic Lists**

The functionality that is offered by graphic lists CAN be obtained, alternately, by storing the desired graphics commands in display lists, and executing the related display list whenever a button-page change command is sent. However, graphic lists offer several advantages over this technique.

First of all, no available auxiliary memory is required; graphic lists can be used even when there is none installed. Commands also execute more quickly from system RAM than from auxiliary memory.

The primary advantage of graphic lists, when linked to button pages, is that the set of graphics operations is directly linked to the page; the user doesn't need to separately think about drawing graphics at all - it goes with the button set. For example, when the Erase Screen command is sent to the terminal, the current button page is automatically redrawn. A display list would not automatically be run, because there is no relationship between the two operations. A linked graphic list, however, WOULD be drawn when the button page was drawn. This linking allows the user to think of the set of buttons (the button page) and the set of graphics operations as one single unit, and not have to consider them separately at all.

## 9.0 Graphic Objects

The graphic objects in the Quick C9 are a set of composite graphic items which let the user create complex graphical displays with less programming effort. These graphic objects are divided into two conceptual groups; the *passive* objects are simply 3D constructs which can be drawn, along with other graphics and objects, to create more interesting displays. The *active* graphic objects actually allow the user to define an object on the screen that can display changing, real-world data without having to do any screen computations at all!!

### 9.1 Passive Graphic Objects

The *passive* objects (3D areas and separators) can be drawn in combination with other graphics and objects to create more complex and sophisticated user-interface designs. Any of these objects could be created manually by a user, but as graphic objects they are more easily and intuitively utilized.

A 3D Area appears like a graphics-mode 3D button with no label, and can be drawn raised above the surface or recessed below it. 3D areas can be used to group other objects such as buttons together, or can be used to emphasize messages and other display information.

A 3D separator is a horizontal or vertical line which can also be either raised or sunk. They can be used to enhance the appearance of graphic designs by highlighting various elements.

The following example creates a row of three buttons, then uses passive graphic objects to make it appear as a labeled toolbar. It is located on the Utilities & Demos disk as TOOLBAR1.TXT.

```
; Notice in this example that only the buttons and the
; initial graphic object are drawn with absolute coordinates.
; All other objects are drawn with Relative draw commands.
; This means that the entire group of graphics items can be
; moved simply by changing the first absolute draw position!!

; define raised 3D Area for toolbar
DCS 21G1;0;220;70;1 ST

; define sunken horizontal 3D Separator
DCS 21G2;1;220;0;0 ST

; Draw 3D Area, Absolute
DCS 23G1;390;60 ST

; draw label on toolbar.
; Note the use of Push/Pop Graphic Cursor Position commands,
; to restore the graphic cursor to a known position after
; the string draw.
DCS 37G1 ST
DCS 2G-19;50 ST
DCS 12GA Deeco Toolbar ST
DCS 37G ST

; move the graphic cursor down and then:
; Draw 3D Separator, Relative
DCS 2G-21;0 ST
DCS 23G2 ST
```

```

; define and draw the row of three buttons
CSI > 1;40;10;44;17B
CSI > 2;40;19;44;26B
CSI > 3;40;28;44;35B
DCS ~1/|:File ST
DCS ~2/|:Edit ST
DCS ~3/|:View ST

```

Download this example to the terminal, and observe how 3D objects not only make the presentation more interesting, but also clearly demonstrate how the buttons are related to each other.

## 9.2 Active Graphic Objects

The *active* graphic objects (Gauges and Trend Graphs) give the user a tool for displaying real-world data, which varies over time, without having to translate it to display values in any way!! Imagine being able to say "I want to display the conveyor-belt speed on a bar graph, and the belt speed varies between 0-250 (meters/second). I'll tell the module what speed the belt is currently at, and I want IT to figure out how to draw the graph correctly.". That is exactly how active graphic objects work. Initially, the user defines the object, telling it what the minimum and maximum data values are for their input data. After that, the user sends an Update Graphic Object command, with the new data value *in user coordinates*, and the Quick C9 automatically redraws the graph to reflect the new value. All screen coordinates are calculated, quickly and silently, by the terminal!!

There are four types of active graphic objects available on the Quick C9:

- Horizontal Bar Graphs
- Vertical Bar Graphs
- Meters
- Trend Graphs

Later sections of this chapter will discuss these objects in more detail.

The following example demonstrates a horizontal bar graph, and then uses the passive graphic objects to enhance the graph and make it more visually compelling.

### **Example 1: a simple horizontal bar graph**

This example is available as file HBG1.TXT. It creates a plain Horizontal Bar Gauge graphic object with a separate label drawn above it, draws the object, then shows how to update it.

```

; create gauge. The range of input data is 500-800 units
DCS 21G 4; 2; 151; 58; 1; 500; 800 ST

; draw the gauge
DCS 23G 4; 200; 100 ST

; update the gauge once, so it displays an initial value
DCS 22G 4; 550 ST

; draw a label above the gauge
DCS 12G 225; 96A fuel pump pressure ST

```

Now, send the following commands to the module, one at a time, and observe the changes in the display. Note: these commands are stored in a file called HBGU1.TXT; in that file, the commands are separated by a code which (assuming that HOST is used to download the file) will pause after each line is sent, waiting until a key is pressed on the PC keyboard before sending the next line.

```

; update the object with different values,
; pausing after each command.
DCS 22G 4;550 ST
DCS 22G 4;610 ST
DCS 22G 4;640 ST
DCS 22G 4;690 ST
DCS 22G 4;720 ST
DCS 22G 4;760 ST
DCS 22G 4;800 ST
DCS 22G 4;610 ST

```

Okay, that created a bar graph, and it works as predicted. However, it could certainly be presented in a more interesting and informative manner. The next example will create the same graph, but will use passive graphic objects and some other labeling, to give it a more professional presentation.

**Example 2: an enhanced horizontal bar graph**

This example is available as HBG2.TXT.

```

; create 3D Area
DCS 21G1;0;100;200;1 ST

; create separator
DCS 21G2;1;200;0;0 ST

; create separator
DCS 21G3;1;77;1;1 ST

; create gauge
DCS 21G4;2;151;58;1;500;800 ST

; create separator
DCS 21G5;1;200;0;1 ST
CSI >5;5;2 O

;*****
; Draw 3D Area, Absolute
DCS 23G1;405;45 ST

; Graphic string draw for label
DCS 36G2;11;1 ST
DCS 37G1 ST
DCS 2G-19;20 ST
DCS 12GA fuel pump pressure ST
DCS 37G ST
DCS 2G-21;0 ST

; draw horizontal separator
DCS 23G2 ST

```

```

; draw max label
DCS 36G2;1;7 ST
DCS 37G1 ST
DCS 2G-19;0 ST
DCS 12GA 800 - ST
DCS 37G ST

; draw min label
DCS 37G1 ST
DCS 2G-77;0 ST
DCS 12GA 500 - ST
DCS 37G ST

; draw vertical separator
DCS 2G0;46 ST
DCS 23G3 ST

; draw and update graphic object
DCS 2G-10;0 ST
DCS 23G4 ST
DCS 22G4;600 ST

```

Now, send HBGU1.TXT to the module again. The graph operates properly, just like the previous example, but communicates much more effectively with the observer.

### 9.3 Working with Graphic Objects

There are up to four stages to creating and using graphic objects:

- Define Graphic Object  
This command specifies an ID number and object type for the graphic object, and provides other basic parameters for specifying it. Each type of graphic object has different additional parameters. See the Command Reference for detailed information on defining each type of graphic object:  
  - ⇒ Define 3D Area (type 0)
  - ⇒ Define 3D Separator (type 1)
  - ⇒ Define Gauge (type 2)
  - ⇒ Define Trend Graph (type 3)
- Define Graphic Object Attributes (optional)  
This command allows the user to change various display characteristics of a graphic object, such as colors, width and height, whether it is raised or sunken, and various water marks. Note that a separate command, Set Graphic Object Marks, provides more control over water marks. These special attributes are discussed separately in this manual.
- Draw Graphic Object  
This command draws a previously-defined graphic object on the screen.
- Update Graphic Object  
Once an active graphic object is drawn, the value displayed on it can be changed using the *Update Graphic Object* command. Remember that Graphic Objects allow users to display data in the *real-world values* that they actually use!! All translation of data into display coordinates is handled internally by the Quick C9.

The Utilities and Demos disk contains a ZIP file called GOBJECTS.ZIP, which contains an interactive demonstration of all of the graphic objects. Extract the files from GOBJECTS.ZIP onto a disk drive, and run the batch file GODEMO.BAT to download and run the graphic object demo.

## 9.4 Working with Trend Graphs

The Trend Graph is a special type of active graphic object which operates differently than the other types. Whereas gauges and meters display a single value within a specified range, the trend graph displays a set of data points on a line graph. When the graph fills up it begins scrolling to the left, discarding the oldest data points and replacing them with newer points.

## 9.5 Linking Trend Graphs

In many cases, one may wish to display two or more related plots on one graph, to show relationships between data. The Quick C9 allows you to link multiple trend graphs together so that they plot on the same graph. The display coordinates of the first graph in the list are used to plot all of the linked graphs. An additional advantage of linking graphs is that ONE SINGLE update command is used to update ALL of the linked graphs. This makes updates even faster than updating several separate graphs. See *Link Trend Graph*, *Unlink Trend Graph*, and *Clear Trend Graph* commands in the Command Reference for details on linking trend graphs.

## 9.6 Water Marks and Control Responses

- High-water and Low-water marks  
Each of the active graphic objects can have a high-water mark and/or a low-water mark defined for it. These water marks are specified as user's real-world data values, as are all data values for active graphic objects. Water marks are displayed as lines on the specified graphic object, with the colors specified by the programmer. Water marks can also be used to trigger control responses, as discussed in the next section.
- Control Responses  
High and Low water marks can have host and/or local responses associated with them, just as buttons and menu items do. The responses are executed when the selected event occurs. The defined events are:
  - ⇒ Entering high-water region
  - ⇒ Leaving high-water region
  - ⇒ Entering low-water region
  - ⇒ Leaving low-water region

The *Set Graphic Object Marks* and *Define Control Responses* commands are documented in the Command Reference chapter.

- Maximum/Minimum marks  
These water marks are displayed just like high/low water marks, except these marks simply show the highest and lowest levels that the data on a graph has attained since they were turned on. There are no control responses associated with max/min marks.

## 9.7 Graphic Object Report

The *Report Graphic Objects* command requests a text listing of all currently-defined graphic objects. This report is sent to the host via the serial port.



## 10.0 Miscellaneous Topics

### 10.1 Adjusting Touch Screen Sensitivity

The touch screen sensitivity may be adjusted for harsh environments to exclude any touch which might be considered too small or too large. Additionally, a delay can be added to require more than just a quick hit.

This feature reduces undesired actuations or so-called "false hits". A false hit might be caused by objects laying on, dropped onto or bounced off of the touch panel, or by the touch panel being bumped or brushed by a person, an animal or an insect.

To avoid accidental touch reports, the minimum and maximum size of objects that will trigger the touch sensor may be programmed. A delay between the time a touch is sensed and the time the report is triggered may also be programmed.

To program the touch sensitivity, send the command:

```
CSI>Pmax; Pmin; Pd S
```

Where:

**Pmax** is the maximum-sized object that will trigger the touch sensor. Range 2-80. Each increment equals .192 inches (4.88 mm). The maximum size must always be at least two increments larger than the minimum.

**Pmin** is the minimum-sized object that will trigger the touch sensor. Range 0-78 Each increment equals .192 inches (4.88 mm).

**Pd** is the delay in increments of 0.04 seconds. Range 0-50.

#### Examples:

```
CSI > 10; 3; 35 S
```

Maximum size for valid touch is 1.92 inches (48.8 mm), minimum size for valid touch is .576 inches (14.6 mm); delay is 1.4 seconds.

```
CSI > 3; 0; 0 S
```

Maximum size is .576 inches (14.6 mm), minimum size is undefined; there is no delay.

**NOTE:** Spaces shown in command are for clarity only. No spaces are inserted when command is sent.

### 10.2 Self-Check Functions

#### 10.2.1 System Operation Commands

The Self-Check command will check the operation of all infrared beams, test the system ram, and check the program memory checksum. This test uses a special technique to test all ram while not disturbing the contents. This takes a little longer than a simpler destructive memory test, but leaves all state information, communication queues, and button definitions intact. This will take either four seconds, or sixteen seconds, depending on the size of the ram chip installed.

During this period, the device will not respond to touches or computer commands (and none should be attempted or they will be lost).

**The screen should not be touched during the test or erroneous results for the I-R beams will result.**

The Self-Check command is as follows:

```
CSI > 4 n
```

The sensor will return the results in the following format:

```
CSI > Pf c
```

Where:

**Pf** is a decimal ASCII digit which will be 0 if all systems are working properly. The low order bits each represent one of the systems.

- 1 Checksum failure
- 2 Ram test failure
- 4 I-R beam failure

Each error found by the self-test will increment an error counter which may be interrogated and cleared with the command:

```
CSI > 2 n
```

Which will respond with:

```
CSI > Pn e
```

Where **Pn** is a decimal number indicating the error count: Range = 0 to 65535

The module has the ability to send a beam failure report on request. The command is:

```
CSI > 3n
```

The module will respond with:

```
CSI > Pf ; Pf ; Pf ; . . . Pf a
```

Where **Pf** is a decimal ASCII integer that identifies which component has failed. Multiple failure codes may be sent. Component identification codes begin with 01 for the leftmost LED on the X axis. Even numbers identify the LEDs and odd numbers identify the photo-transistors. Numbers count upward to the right in the X axis, and upward in the Y axis. There is no gap in the numbering between the axes.

## 10.2.2 Configuration Inquiry

The touch sensor supports two inquiries for determining its configuration. The dimensions of the touch array may be requested by sending:

```
CSI > 8 n
```

This will return:

```
CSI > Py; Px d
```

Where:

**Py** is a decimal ASCII integer which is the maximum Y axis coordinate which can be returned by the sensor. (Ymax)

**Px** is a decimal ASCII integer which is the maximum X axis coordinate which can be returned by the sensor. (Xmax)

Coordinates returned by the sensor range from one to the maximums returned by this command.

The current version of the software may be requested by sending:

```
CSI > 9 n
```

Which will return:

```
CSI > Pv v
```

Pv is a decimal ASCII integer which reflects the version. The number 10 implies 1.0.

### 10.2.3 List-reporting Commands

A series of reporting commands are now provided in the C9. Each of these commands sends a full text report to the host, containing all pertinent information about the requested group of objects. For example, Get Button List Report returns a listing of all button pages, including all buttons on each page, showing their positions, sizes, attributes, responses and labels. The new report commands are:

```
Get Button List Report      CSI > 11 n
Get Auxiliary Memory List Report CSI > 12 n
Get Menu List Report       CSI > 13 n
Get Keyboard List Report   CSI > 14 n
Get Graphics List Report   CSI > 15 n
```

This is an example of a report received in response to *Get Auxiliary Memory List Report*:

```
current Auxmem list report
=====
Display list 0, 9 bytes, csum=good
Font list    1, 4256 bytes, csum=good
Font list    2, 4256 bytes, csum=good
Font list    3, 4256 bytes, csum=good

End of Report...
```

## 10.3 The Real-Time Clock

The Quick C9 now contains a real-time clock which can be displayed on the screen. Any combination of date, time, and seconds can be displayed, and position and colors of the display may be chosen by the user. The commands for setting the time, screen location, and other attributes are documented in the Command Reference.

## 10.4 Auto-Configure

The controller may be directed to configure itself automatically upon power-up to a user-defined state. The state refers to various modes of operation which are selectable using host commands.

Normally, the controller power-up state is the normal default state. **Auto-configure requires auxiliary memory.**

In auto-configure, the controller is directed to execute display list #0 upon power-up. This display list may contain commands to alter the normal default state.

For the C9, auto-configure is controlled by jumpers E15 and E16 on the controller board, or by the set-up screen. For normal default state, no jumpers are installed in these positions. To direct the controller to

execute display list #0 upon power-up, a jumper is placed in E15. Auto-configure can also be selected in the set-up screen.

## 10.5 Set-Reset Functions

The Quick C9 provides a group of functions which are generically referred to as *Set/Reset* functions. These functions begin with `CSI` and traditionally end with either `h` to SET a value or `l` to CLEAR the value. For example, The *Set Button System Options* comprise a series of commands from `CSI > 23 h/l` through `CSI > 28 h/l`. Thus, to enable highlighted button touches, the host would send `CSI > 26 h` to the C9.

In the Quick C9, the Set/Reset functions have been augmented with two more options, called *toggle* mode and *query* mode. To access *toggle* mode, use `j` in place of `h/l`, and to access *query* mode, use `#` in place of `h/l`. When a query command is sent to the module, it responds with the normal Set/Reset form of the command.

### Example 8.5a

The normal command for *Enable Command Error Reporting* is `CSI > 34 h`. To toggle the state of this value, use the *Toggle Command Error Reporting* command, which is `CSI > 34 j`.

### Example 8.5b

To determine the current state of the *Command Error Reporting* flag, send the *Query Command Error Report* command, which is `CSI > 34 #`. The module will respond with `CSI > 34 h` if the flag is enabled, or `CSI > 34 l` if it is disabled.

The *Toggle* and *Query* command forms can be used with any of the *Set/Reset* commands. Remember, these are any command beginning with `CSI` and ending with `h` or `l`.

## 10.6 Periodic Functions

*Periodic Functions* provide a mechanism for executing a certain task or tasks at a fixed interval. A list of one or more display lists is created, and communicated to the module. Each time the specified time period elapses, the next display list in the sequence is executed. When the last list is executed, the function loops back to the first. This feature can be used for such tasks as cycling through a list of bitmap images or displayed messages. The file PER\_FCNS.TXT on the Utilities and Demos disk provides an example of a periodic function in use.

## APPENDIX A: C9 Host Command Reference

### A.1 C9 Host Command Reference, by category

Use this section to find the name of a command, so you can look it up in the alphabetic reference.

#### A.1.1 Button Functions

- Define Button System Attribute
- Set Button System Options
- Clear Scrolling Window

- Define Button Area
- Define Text Button Area
- Define Button Attributes, Full Form
- Define Button Attributes, Single Form
- Define Button Responses
- Copy Button Responses
- Get Button Bitmap Status
- Delete Button Bitmap Label
- Copy Button Bitmap Label

- Set Button Page
- Get Button Page Numbers
- Draw Button Page
- Undraw Button Page

- Set Active Button
- Get Active Button
- Raise Button To Top
- Activate Button
- Set Button State
- Draw Button
- Undraw Button

- Delete All Buttons
- Delete Button (All Button Pages)
- Delete Button (Definition Button Page Only)
- Delete Button Page
- Set Button Timeout

- Get Button Free Space
- Get Button Numbers
- Get Next Button Page
- Request Button-List Report
- Get Button List Report

## **A.1.2 Menu Commands**

- Define Menu Responses
- Define Menu Attribute, Single Form
- Define Menu Attribute, Multiple Form
- Define Menu Item Attributes
- Draw Menu
- Undraw Menu
- Undraw Menu
- Undraw All Menus
- Erase All Menus
- Erase Menus
- Erase Menu
- Erase Menu Item
- Get Menu List Report

## **A.1.3 Keyboard Functions**

- Define Key Attributes, Multiple Form
- Define Key Attributes, Single Form
- Define Keyboard Attributes, Multiple Form
- Define Keyboard Attributes, Single Form
- Define Keyboard Responses
- Delete All Keyboards
- Draw Keyboard
- Delete Keyboard
- Lock Keyboard
- Undraw Keyboard
- Unlock Keyboard
- Keyboard Enable/Disable
- Keyboard On/Off
- Set-Up Key Enable/Disable
- Get Keyboard List Report
- Get Current Drawn Keyboard

## **A.1.4 Graphics Drawing Functions**

- Define Graphic Clipping Window
- Graphic Clipping Window, Enable/Disable
- Get Graphic Cursor Position
- Set Graphic Cursor Position, Absolute/Relative
- Arc Draw, Absolute/Relative
- Circle Draw, Absolute/Relative
- Circle Draw, Absolute/Relative With Autofill
- Ellipse Draw, Relative
- Rectangle Draw, Absolute/Relative
- Rectangle Draw, Absolute/Relative With Autofill
- Vector Draw, Absolute/Relative
- Vector Draw Polar, Absolute/Relative
- Vector Draw Polar, Absolute/Relative, Partial
- Vector Draw To
- Block Draw Absolute/Relative, Autofill
- Pattern Fill Absolute/Relative

Graphic String Draw, Absolute/Relative  
Write Graphic Pixels  
Read Graphic Pixels

Graph Cursor Enable/Disable  
Set Graphics Character Attributes  
Set Graphic Cursor Type  
Set Graphic Write Mode  
Set Graphics String Attribute  
Get Fill Style Attribute  
Set Fill Style Attribute  
Get Line Style Attribute  
Set Line Style Attribute  
Set User-Defined Line Style  
Set User-Defined Fill Pattern  
Reset Graphics Character Attributes  
Reset Graphics String Attributes  
Reset Graphics Write Mode

Define Bitmap List  
Draw Bitmap List, Relative  
Draw Bitmap List, Absolute  
Draw Bitmap, Relative  
Draw Bitmap, Relative, Corrected  
Draw Bitmap, Absolute  
Draw Bitmap, Absolute, Corrected  
Define Button Bitmap Label  
Bitmap List Tile  
Bitmap List Fill, Absolute  
Bitmap List Fill, Relative

Enter Fast Vector Mode

Flash Area, Define  
Flash Area, Start  
Flash Area, Stop

Push Graphic Cursor Position  
Pop Graphic Cursor Position

### **A.1.5 Graphic List Functions**

Graphic List Open  
Graphic List Close  
Graphic List Executive, Relative/Absolute  
Graphic List Delete  
Graphic List Link  
Graphic Lists Report

## **A.1.6 Graphic Objects**

- Define 3D Area
- Define 3D Separator
- Define Gauge
- Define Trend Graph
- Update Graphic Object
- Draw Graphic Object, Absolute
- Draw Graphic Object, Relative
- Link Trend Graph
- Unlink Trend Graph
- Clear Trend Graph
- Delete Graphic Object
- Report Graphic Objects
- Define Control Responses
- Set Graphic Object Marks
- Define Graphic Object Attributes, Single Form
- Define Graphic Object Attributes, Multiple Form

## **A.1.7 Touch Functions**

- Report Enter Touches, Enable/Disable
- Report Exit Touches, Enable/Disable
- Report Multiple Touches, Enable/Disable
- Report Tracking Touches, Enable/Disable
- Repeat Previous Touch Report
- Reset Touch Modes

- Highlight Screen Touch
- Get Touch Position

- Beep On Screen Touch
- Auto Repeat On/Off
- Set Bad Beam Beep Time
- Set Bad Beam Report Enable/Disable
- Get Bad Beam List

- Set Touch Screen Sensitivity

## **A.1.8 Screen Functions**

- Get Video Mode
- Set Video Mode
- Get Video Page
- Set Video Page

- Get Color Attributes
- Set Color Attributes
- Get Character Set
- Set Character Set

Text Underline Enable/Disable

Set Text Attributes:

*All Attributes Off*

*Text Underline On*

*Text Underline Off*

*Text Blink On*

*Text Blink Off*

*Reverse Video On*

*Reverse Video Off*

*Italic On*

*Italic Off*

Set Text Blink Rate

Erase Screen

Erase Screen To Cursor

Erase Line To Cursor

Erase Line

Erase Line From Cursor

Erase Screen From Cursor

Normal Video (Screen)

Reverse Video (Screen)

Define Character Bitmap

Set Character Size

Set Text Cursor Style

Set Alpha Write Mode

## **A.1.9 Alpha Cursor Movement Commands**

Text Cursor On/Off

Get Text Cursor Position

Write Text String

Cursor Down

Cursor Home

Cursor Left

Cursor Right

Cursor Set Position

Cursor Up

Next Line

Lf After Cr, Enable/Disable (Alpha Newline)

Horiz Tab Clear

Horiz Tab Set

Horiz Tab Clear All

Index

Reverse Index

### **A.1.10 Printer Functions**

- Print Control Off
- Print Control On
- Auto Print Off
- Auto Print On
- Get Printer Status
- Print Buffer Request
- Print Buffer Response
- Print Cursor Line
- Print Formfeed, Enable/Disable
- Print Screen
- Printer Input Buffering, Enable/Disable
- Printer Input Enable/Disable
- Printer Port Configure

### **A.1.11 Auxiliary Memory List Functions**

- Display List Close
- Display/Data List Open
- Display List Clear All
- Display List Compute/Store Checksum
- Display List Execute
- Display/Data List Append
- Display/Data List Read
- Display/Data List Get Information

- Delete Auxiliary Memory List
- Get Auxiliary Memory List Report
- Get Display List Free Memory

- Periodic Function Enable/Disable

- Store Font List
- Load Font List Into Character Set
- Correct Corrupted Auxmem Lists

### **A.1.12 Miscellaneous Functions**

- Reset To Initial State

- Query Set/Reset State
- Toggle Set/Reset State
- Transmit 7-Bit Control-Code Sequences
- Transmit 8-Bit Control Codes
- Transmit 8-bit, request state
- Enable 8-Bit Control
- Enable 8-Bit Data

- Get Device Status
- Get Firmware Version
- Auxiliary Memory Checksum Request

- Screen Saver On/Off
- Screen Saver Time-Out
- Screen Saver Reset On Receive, Enable/Disable

Multidrop Enable/Disable  
Multidrop Receive Control  
Multidrop Transmit Control, Continuous  
Multidrop Transmit Control, Short-Span  
XON/XOFF Enabled/Disabled  
Enter Selftest Mode

Module State Save  
Module State Restore  
Module State Default

Get Button List Report  
Get Auxiliary Memory List Report  
Get Menu List Report  
Get Display Information

Define Auto-Execute Mode  
Sleep

### **A.1.13 Error Reporting Functions**

Command Error Reporting, Enable/Disable  
System Status Request  
System Error Reporting, Active/Passive  
Read System Error Counters  
System Self-Test  
Get Selftest Error Counter

### **A.1.14 Real-Time Clock (RTC) Commands**

Set RTC Time  
Set RTC Display Format  
Set RTC Attributes, Multiple Form  
Set RTC Attributes, Single Form



## A.2 C9 Host Command Reference, Alphabetical listing

ACTIVATE BUTTON: CSI > Pb A

Activate a button from the host, rather than by touch.

Pb: Button number.

ALPHA CURSOR ON/OFF: CSI ? 25 h/l

Turn the alpha cursor on or off.

ALPHA NEWLINE (LF AFTER CR), ENABLE/DISABLE: CSI 20 h/l

If this function is enabled (CSI 20 h), a linefeed (LF) appended to each carriage return (CR) (or enter key, when the numeric keypad is in numeric mode), and a CR is appended after each LF.

ARC DRAW ABSOLUTE: DCS 8 G Pa1; Pa2; Pr; Py; Px ST

ARC DRAW RELATIVE: DCS 8 G Pa1; Pa2; Pr ST

Draws an arc on the screen centered on the reference point, which is (Py;Px) (absolute) or the CP (relative). The arc is drawn between angle Pa1 and angle Pa2, with radius Pr. All angles are in degrees in the Quick C9. The CP is not changed by this command.

It should be noted that arcs on the C9 are *partial circles*, not true arcs.

AUTO PRINT ON: CSI ? 5 i

AUTO PRINT OFF: CSI ? 4 i

Enable or Disable auto-print mode. Auto-print mode causes a line to be printed when the cursor advances to the next line due to a linefeed, formfeed, vertical tab, or autowrap (CR, LF). The printed line ends with a carriage return and the LF, FF, or VT.

AUTO REPEAT ON/OFF: CSI ? 8 h/l

Enabling auto-repeat cause a key depressed more than 0.5 seconds to be repeated, except for these keys: HOLD SCREEN, PRINT SCREEN, BREAK, RETURN, LOCK, SHIFT and CONTROL.

AUXILIARY MEMORY CHECKSUM REQUEST: CSI < 2 n

Request for report on the auxiliary memory checksum test.

Report format:

CSI < 2; Pr; Ps; Pc n

Function: Response to report request of the auxiliary memory checksum test. The result of the checksum test is reported with two numbers, the checksum and the result of the checksum match. The auxiliary memory checksum is stored in the first byte of the auxiliary memory device. The checksum is the module 256 (8-bit) sum of all bytes in the memory except the first byte. The checksum reported is the calculated value, the result reported is 1 if that value is stored in the first memory location and 0 if it is not.

Pr: Result (0 or 1) of the checksum test.

Ps: Size of auxiliary memory; 0=none, 1=8K, 2=32K, 3=64K, 4=128K

Pc: Calculated checksum.

BEEP ON SCREEN TOUCH: CSI > 21 h/l  
Generate a beep when the screen is touched. This will generate a beep when any button is touched which has C9 reports enabled, including the background button0.

BITMAP LIST TILE: DCS 24 G Pid; y1; x1; yu; xr ST  
This command functions exactly like Block Draw, except that it fills the specified region by tiling Bitmap List Pid, rather than using a C9 fill pattern.  
Pid: Bitmap List number.

BITMAP LIST FILL, ABSOLUTE: DCS 35 G Pid; Py; Px ST  
BITMAP LIST FILL, RELATIVE: DCS 35 G Pid ST  
This command functions exactly like Pattern Fill, except that it fills the specified region with Bitmap List Pid, rather than using a C9 fill pattern.  
Pid: Bitmap List number.

BLOCK DRAW RELATIVE: DCS 9 G Py2; Px2 ST  
BLOCK DRAW RELATIVE, WITH AUTOFILL: DCS 9 G Py2; Px2; Pf ST  
BLOCK DRAW ABSOLUTE: DCS 9 G Py1; Px1; Py2; Px2 ST  
BLOCK DRAW ABSOLUTE, WITH AUTOFILL: DCS 9 G Py1; Px1; Py2; Px2; Pf ST  
Fills a rectangle on the screen from (Py1;Px1) (absolute) or from the CP (relative) to point (Py2;Px2) with the default fill pattern. This is similar to calling Rectangle Draw with AutoFill, except no border is drawn. The CP is set to (Y2,X2) after the operation. If fill pattern Pf is provided, the block is fill with the requested pattern, rather than with the default fill pattern. Use *Bitmap List File* to fill a rectangle with a tiled bitmap.

CIRCLE DRAW RELATIVE: DCS 7 G Pr ST  
CIRCLE DRAW RELATIVE, WITH AUTOFILL: DCS 7 G Pr; Pf ST  
CIRCLE DRAW ABSOLUTE: DCS 7 G Pr; Py; Px ST  
CIRCLE DRAW ABSOLUTE, WITH AUTOFILL: DCS 7 G Pr; Py; Px; Pf ST  
Draws a circle on the screen centered on the reference point, which is (Py;Px) (absolute) or the CP (relative), with radius Pr. If fill pattern Pf is provided, the rectangle is filled with the requested pattern, using the Pattern Fill function. The CP is not changed by this command.

CLEAR GRAPHICS CLIPPING WINDOW: DCS 0 G ST  
Erases the region bounded by the graphics clipping window, using the current Graphics Object Background color. If graphics clipping is disabled, the entire definition video page is cleared. In text mode, this calls *Erase Screen*.

CLEAR SCROLLING WINDOW: CSI < e  
Erase the contents of the active window, and homes the cursor.

CLEAR TREND GRAPH: DCS 22 G Pid ST  
This command clears all previously-entered data from the data queue of a trend graph. If Pid is part of a set of linked trend graphs, ALL of the linked graphs are cleared.

COMMAND ERROR REPORTING, ENABLE/DISABLE: CSI > 34 h/l  
Enable (h) or disable (l) command-error reporting. When this is enabled, and the terminal receives any commands which have incorrect syntax or arguments, it sends an error report back to the host. See the discussion of command error reporting, elsewhere in this manual, for a detailed discussion of the format of these error reports.

**COPY BUTTON BITMAP LABEL:**

**CSI > db; sb; sbp ST**

Copies a button bitmap label from source button **sb** on button page **sbp**, to destination button **db** on the definition button page. Both source and destination buttons must already be defined. If the destination button is on the current button page, the button will **not** be automatically redrawn; a *Draw Button* command, or equivalent command such as *Erase Screen*, must be sent to redraw the button with the new button label.

**COPY BUTTON RESPONSES:**

**CSI > Pnb ; Pob ; Pop C**

Copy the button responses from one button to another. The new button is always on the definition button page, the existing button can be on any existing button page. The entire button state list, including colors, responses, labels, etc are copied; this command cannot be used to copy only one state from a multi-state button, for example.

This command offers an effective method for conserving memory when the same button definition is used on multiple button pages, since it does not actually copy the responses themselves, it merely creates a second pointer to the existing button state list.

- Pnb:** Button number for new (destination) button.
- Pob:** Button number for old (source) button.
- Pop:** Button page number for old (source) button.

**CORRECT CORRUPTED AUXMEM LISTS:**

**CSI < -1 C**

Cleans up any corrupted pointers in auxiliary memory, so that subsequent *Delete Auxiliary Memory List* commands can be used to repair errors in the auxiliary memory.

**CURSOR DOWN:**

**CSI Pn B**

Move the cursor down. The cursor will stop at the bottom of the scroll region.  
**Pn:** Number of lines to move down.

**CURSOR HOME:**

**CSI H**

Returns the text cursor to line 1, column 1. The contents of the screen are not changed.

**CURSOR LEFT:**

**CSI Pn D**

Move the cursor left. The cursor will stop at the left margin.  
**Pn:** Number of character positions to move.

**CURSOR RIGHT:**

**CSI Pn C**

Move the cursor right. The cursor will stop at the right margin.  
**Pn:** Number of character positions to move.

**CURSOR SET POSITION:**

**CSI P1; Pc H**

Move the cursor to the specified character line and column. Rows are numbered 1 through 25, columns from 1 to 80. Row or column 0 is interpreted as row or column 1. Default for a missing parameter is 1. If the row or column specified is greater than the actual number, then the largest number possible is used.

- P1:** Line number.
- Pc:** Column number.

**CURSOR UP:**

**CSI Pn A**

Move the cursor up. The cursor will stop at the top of the scroll region.  
**Pn:** Number of lines to move up.

DEFINE 3D AREA (TYPE = 0): DCS 21 G Pid; 0; Pw; Ph; Pr ST

The 3D Area is one of the set of graphic objects. It is a square region of the screen with a 3D border around it.

- Pid:** Graphic object number for this object. Range of Pid is 0-65000.
- Pw:** Width of region in pixels (not including border)
- Ph:** Height of region in pixels (not including border)
- Pr:** *Raised* flag. If this is 0, the area is drawn sunken into surface, like a button that is pressed; if this is 1, the area is drawn as a raised.

DEFINE 3D SEPARATOR (TYPE = 1): DCS 21 G Pid; 1; Pl; Pv; Pr ST

The 3D Separator is one of the set of graphic objects. It is horizontal or vertical line which appears to have height (or depth).

- Pid:** Graphic object number for this object. Range of Pid is 0-65000.
- Pl:** Length of region in pixels
- Pv:** *Vertical* flag. 0 = draw horizontal line; 1 = draw vertical line.
- Pr:** *Raised* flag. If this is 0, the line is drawn sunken into surface; if this is 1, the line is drawn as a raised line.

DEFINE AUTO-EXECUTE MODE: CSI < Pa A

Select the operating mode of the module at boot time.

- Pa:** operating mode
  - 0 = Normal operation
  - 1 = Auto-execute display list 0
  - 2 = Enter self-test mode

Note that in order for this command to take effect, a save-state operation must be executed and then the unit must be rebooted. See the documentation for the *Module State Save* command for more information.

Note that self-test can be entered immediately by using the *Enter Self-test Mode* command **CSI ! t**.

DEFINE BITMAP LIST: DCS 29 G Pi; Pw; Pc H <data> ST

The *Define Bitmap List* command is used to store a bit-mapped image in auxiliary memory as Bitmap List **Pi**. Parameter **Pw** is the width of the bitmap in pixels. **Pc** is the *color depth* flag: 0 for 16-color images, or 1 for 256-color images. Once a Bitmap List is stored, it can be drawn using the *Draw Bitmap List* command. The Bitmap is drawn from bottom to top, in the same manner as button bitmap labels are drawn. Because the list is stored as binary data, *Draw Bitmap List* draws MUCH more quickly than the related *Draw Bitmap* command can. However, unlike images drawn with *Draw Bitmap*, Bitmap List image size is limited by the amount of available auxiliary memory, and in any case is limited to 64000 bytes per object.

Note that for 16-color images, the bitmap data contains 1 byte/pixel, while for 256-color images, the bitmap data contains 2 bytes/pixel, MSB first.

DEFINE BUTTON AREA: CSI Pb ; Y1 ; X1 ; Yu ; Xr B

Define the area for a button. This command defines button boundaries using touch-system coordinates (Y = 1-49, X = 1-80).

- Pb:** Button Number.
- Y1:** Lower row number.
- X1:** Left column number.
- Yu:** Upper row number.
- Xr:** Right column number.

DEFINE BUTTON ATTRIBUTES, FULL FORM: `CSI > Pb; Ps; Pi; Pi; . . . Pi F`  
 Set ALL button attributes for one button state.

DEFINE BUTTON ATTRIBUTES, SINGLE FORM: `CSI > Pb; Ps; Pi; Pv F`  
 Set ONE button attribute for one button state.

- Pb:** button number on definition page
- Ps:** button state (single or multiple state)
- Pi:** button attribute item number (see table below)
- Pv:** value to assign to button attribute

Button Attributes (button-state specific)		
Attrib Number	Text-mode Usage	Graphics-mode Usage
0	Border Foreground	Border (lit side)
1	Border Background	Border (shadow side)
2	Border Highlighted Foreground	Border pattern-fill Foreground
3	Border Highlighted Background	Border pattern-fill Background
4	Label Foreground	Label Foreground
5	Label Background	Label Background
6	Logo Foreground	Label Highlighted Foreground (for full button redraw)
7	Logo Background	Label Highlighted Background (for full button redraw)
8	(unused)	Label Text Angle (0=horizontal, 1=vertical)
9	(unused)	Label Character Set (0-3)
10	(unused)	Character Size (0-7)
11	(unused)	Fill Pattern (0-11)
12	Border Style	(unused)

Button Attributes (button-specific)		
Pi	Attribute Name	Attribute Description
13	scrolling	0=centered label in button 1=scrolling button text
14	touchable	enable/disable button activation
15	highlight	enable/disable highlighting button on touch
16	c9reports	0=send host/local responses when activated. 1=send global C9 reports when activated.
17	autodraw	enable/disable drawing button when button page is drawn

Button Border Styles	
bstyle	Description of border style
0	Single-line border
1	Double-line border
2	Double horizontal lines, single vertical lines
3	Single horizontal lines, double vertical lines
4	No border; border region is used for text

DEFINE BUTTON BITMAP LABEL: DCS 26 G Pb; dy; dx; Pc H <ddd...> ST  
Assign a bitmap image to existing button Pb. Maximum image size is 65000 pixels.

- Pb:** button number
  - dy:** height of bitmap in pixels
  - dx:** width of bitmap in pixels
  - Pc:** color depth (optional)
    - 0 = 16-color image, with one ASCII-HEX byte per pixel
    - 1 = 256-color image, with two ASCII-HEX bytes per pixel, most significant byte first.
- If Pc is not specified, this flag will be determined by the current video mode of the module; it will be 0 for text or VGA16 video mode, and 1 for VGA256 video mode.

DEFINE BUTTON SYSTEM ATTRIBUTE: CSI > Pi; Pv G  
Set a global button system attribute.

- Pi:** button system attribute item number (see table below)
- Pv:** value to assign to button system attribute

Button System Attributes	
Pi	Description
0	auto draw: enable/disable button-page draw on page or video-mode change
1	button enable: globally enable/disable current button page
2	beep on button touch (0=OFF, 1=ON)
3	highlight button when touched (0=OFF, 1=ON)
4	button enter/exit mode (1=Enter mode, 0=Exit mode)
5	beep on touch report (0=OFF, 1=ON)
6	button border width for graphics buttons, in pixels (default = 5 pixels)
7	highlight border only. 1=highlight border only when touched, 0=redraw entire button when touched.
8	set button repeat time, in 10msec units. 0=disable auto-repeat

DEFINE BUTTON RESPONSES: DCS ~ Pb / Rh : Rl : Rb : Rg ST

This command defines host response, local response, button label, and button logo strings for each state in a button. To define responses for multi-state buttons, follow Rg with a / and the responses for the next state. For a clearer description of this process, see the section on Buttons elsewhere in this manual.

- Pb:** Button Number.
- Rh:** Button host response string.
- Rl:** Button local response string.
- Rb:** Button label string.
- Rg:** Button logo string (text-mode only).

**DEFINE CHARACTER BITMAP:** **DCS 33 G Pf; Pc; P0; P1; . . . Pn ST**  
 Define the bit pattern of an alternate character. The character is defined in an 8x19 pixel array, where n is the character cell height. The character code defines where in the alternate character memory the character pattern will be stored. The 8x19 pattern maps to the normal character image with the first byte sent corresponding to the top of the character, and the LSBs of the bytes corresponding to the left edge of the character.

**Pf:** Character set in which the character is defined (default = 1).  
**Pc:** Character code of the character being defined.  
**P0-Pn:** 8 bits of character image.

**DEFINE CONTROL RESPONSES:** **DCS& Pid /... ST**  
 Define Host and Local responses for high-water and low-water events from active graphic object **Pid**. Before these responses can have an effect, the graphic object must be created, and high/low-water marks must be defined for it, using the *Set Graphic Object Marks* command, documented elsewhere in this file. If high/low water marks are defined for an active graphic object, and one or more of these responses are defined, the appropriate host and local responses will be processed when the event occurs. The full form of the command is:

**DCS & Pid / ehhr: ehlr / xhhr: xhlr  
 / elhr: ellr / xlhr: xllr ST**

where: **Pid:** ID number for graphic object  
**ehhr, ehlr:** host and local responses for *Enter High-water region* event  
**xhhr, xhlr:** host and local responses for *Exit High-water region* event  
**elhr, ellr:** host and local responses for *Enter Low-water region* event  
**xlhr, xllr:** host and local responses for *Exit Low-water region* event

**DEFINE GAUGE (TYPE = 2):** **DCS 21 G Pid; 2; Pw; Ph; Ps; Dn; Dx ST**  
 The Gauge is one of the set of graphic objects. See the discussion in the Graphic Objects chapter for more information on using this object.

**Pid:** Graphic object number for this object. Range of Pid is 0-65000.  
**Pw:** Width of region in pixels (not including border)  
**Ph:** Height of region in pixels (not including border)  
**Ps:** Gauge style  
 0: Horizontal bar gauge  
 1: Vertical bar gauge  
 2: Meter  
**Dn, Dx:** Minimum (**Dn**) and Maximum (**Dx**) values for raw data.

**DEFINE GRAPHICS CLIPPING WINDOW:** **CSI < Py1; Px1; Pyu; Pxr w**  
 Define the rectangular bounds of a graphic window. The bounds are given as relative offsets from the window origin. The rectangle defined by these offsets are used as the clipping rectangle when clipping is enable.

**Py1:** Y coordinate, lower left corner (relative to window origin).  
**Px1:** X coordinate, lower left corner (relative to window origin).  
**Pyu:** Y coordinate, upper right corner (relative to window origin).  
**Pxr:** X coordinate, upper right corner (relative to window origin).

**DEFINE GRAPHIC OBJECT ATTRIBUTES, SINGLE FORM:** **CSI > Pid; index; value 0**  
**DEFINE GRAPHIC OBJECT ATTRIBUTES, MULTIPLE FORM:** **CSI > Pid; value; ... value 0**  
 Defines one or more attributes for graphic object **Pid**. In order to use the multiple form of this command, at least four arguments (**Pid** plus three attributes) must be specified. The sequence of graphic object attributes depends upon the *type* of the graphic object, and is specified in the tables that follow:

<b>3D Area (type 0) attributes</b>	
<b>index</b>	<b>description of attribute</b>
0	Width of 3D area in pixels, excluding border
1	Height of 3D area in pixels, excluding border
2	Raised (1) or Sunk (0)
3	Border color, highlighted
4	Border color, shadow
5	Face foreground color, normal
6	Face background color, normal
7	Face foreground color, highlighted
8	Face background color, highlighted
9	Face fill pattern, normal
10	Face fill pattern, highlighted
11	Border width in pixels

<b>3D Separator (type 1) attributes</b>	
<b>index</b>	<b>description of attribute</b>
0	Length of separator in pixels
1	Draw vertical line (1) or horizontal line (0)
2	Raised (1) or Sunk (0)
3	Border color, highlighted
4	Border color, shadow
5	Line thickness in pixels

<b>Gauge (type 2) attributes</b>	
<b>index</b>	<b>description of attribute</b>
0	Width of 3D area in pixels, excluding border
1	Height of 3D area in pixels, excluding border
2	Gauge style 0 = Horizontal bar gauge 1 = Vertical bar gauge 2 = Meter
3	Border color, highlighted
4	Border color, shadow
5	Face foreground color, normal
6	Face background color, normal
7	Face foreground color, highlighted
8	Face background color, highlighted
9	Face fill pattern, normal
10	Face fill pattern, highlighted
11	High-water mark color
12	Low-water mark color
13	Maximum value mark color
14	Minimum value mark color

Trend Graph (type 3) attributes	
index	description of attribute
0	Width of 3D area in pixels, excluding border
1	Height of 3D area in pixels, excluding border
2	Graph style 0 = Plot points only 1 = Plot solid line from bottom-of-graph to point
3	(unused)
4	(unused)
5	(unused)
6	(unused)
7	Trend-line color
8	Graph color, background
9	(unused)
10	(unused)
11	High-water mark color
12	Low-water mark color
13	Maximum value mark color
14	Minimum value mark color

DEFINE KEY ATTRIBUTES, MULTIPLE FORM:       CSI > Pid ; Kid ; value; ... value K  
Assigns values to three or more attributes of key *Kid* on keyboard *Pid*. The first value is assigned to index 0, and each subsequent value is assigned to the next sequential key attribute.

DEFINE KEY ATTRIBUTES, SINGLE FORM:       CSI > Pid ; Kid ; index ; value K  
Assign a value to one attribute of key *Kid* on keyboard *Pid*.

Index	Description
0	Normal label foreground color
1	Normal label background color
2	Highlighted label foreground color
3	Highlighted label background color

DEFINE KEYBOARD ATTRIBUTES, MULTIPLE FORM:   CSI > Pid ; -1 ; value;... value K  
Assigns values to three or more attributes of keyboard *Pid*. The first value is assigned to index 0, and each subsequent value is assigned to the next sequential keyboard attribute.

Index	Description
0	Keyboard Style 0 = RESPONSE style; process host/local responses 1 = BUFFERED HOST style; handle buffer as host response 2 = BUFFERED LOCAL style; handle buffer as local response
1	border foreground color
2	border background color
3	Maximum buffered string length. This field is unused for response-style keyboards. It cannot exceed the width of the keyboard.
4	Set graphics-mode video page for this keyboard. Use -1 for "current video page", which is also the default value. This value is not used in text mode.
5	"password" mode. 0=OFF (default), 1=ON

DEFINE KEYBOARD ATTRIBUTES, SINGLE FORM: `CSI > Pid ; -1;index;value K`  
 Assigns a value to one attribute of keyboard *Pid*.

DEFINE KEYBOARD RESPONSES:  
`DCS ? Pid ; Py ; Px & responses % width / responses % width ... ST`  
 Define the position and responses for a user-defined keyboard.  
**Pid:** Keyboard number  
**Py, Px:** Upper-left corner location of keyboard, using text coordinates  
**&:** Begin key on new row  
**/:** Begin key on current row  
**%width:** Optional key width. If omitted, width of key = label length + 2

DEFINE MENU ATTRIBUTE, MULTIPLE FORM: `CSI > Pm ; -1;value;...value M`  
 Defines three or more attribute for menu *Pm*. At least three values must be provided to this command in order for it to work properly. The first value is assigned to index 0, and each following attribute is assigned to the next sequential index in the table.

Menu attributes	
index	Description
0	border foreground color (default = button border FG)
1	border background color (default = button border BG)
2	label foreground color (default = button label FG)
3	label background color (default = button label BG)
4	label highlighted foreground color (default = button label highlight FG)
5	label highlighted background color (default = button label highlight BG)
6	label inactive foreground color (default = button border highlight FG)
7	label inactive background color (default = button border highlight BG)
8	label character set (sets all menu items)
9	border style (not implemented)
10	Menu Orientation: 0=horizontal, 1=vertical (default)
11	Inter-character Spacing: specifies minimum number of spaces before and after each menu item. Default = 1 character

DEFINE MENU ATTRIBUTE, SINGLE FORM: `CSI > Pm ; -1 ; index ; value M`  
 This function defines one attribute for menu *Pm*.

DEFINE MENU ITEM ATTRIBUTES: `CSI > Pm ; Pi ; index ; value M`  
 This function defines attributes for menu item *Pi* in menu *Pm*.

Menu Item Attributes	
index	Description
0	Separator: Marks item as a separator line; all responses are disregarded.
1	Inactive: Draws item with Inactive attribute, and item is not highlighted by touch.
2	Character Set



DELETE ALL KEYBOARDS: CSI > K  
Delete all defined keyboards. Note that keyboards 0 (QWERTY), 1 (KEYPAD) and 2 (NUMERIC) are pre-defined keyboards which cannot be deleted or modified by the user.

DELETE AUXILIARY MEMORY LIST: CSI < P1; Pi C  
Deletes a list number **Pi** of type **PI** from the auxiliary memory. The deleted region is marked as UNUSED; adjacent UNUSED regions will be merged, and if the UNUSED region is at the end of defined data, it will be returned to the available auxmem pool.

- PI:** Auxiliary memory list type
  - 0 = Display List
  - 1 = Data List
  - 2 = (unused)
  - 3 = Font List
  - 4 = Bitmap List
- Pi:** List number
  - If **Pi** = -1, ALL lists of type **PI** are deleted.

DELETE BUTTON PAGE: CSI > 2 D  
Deletes all buttons and their responses on definition button page.

DELETE BUTTON (ALL BUTTON PAGES): CSI > Pb B  
Deletes a button and its responses on all button pages.  
**Pb:** Button number.

DELETE BUTTON (DEFINITION BUTTON PAGE ONLY): CSI > Pb C  
Deletes a button and its responses on all button pages.  
**Pb:** Button number.

DELETE GRAPHIC OBJECT: CSI > Pid O  
Delete graphic object **Pid**.

DELETE KEYBOARD: CSI > 1 ; Pid K  
Delete keyboard **Pid**.

DISPLAY/DATA LIST APPEND: DCS Pid a <h...h> ST  
Open a data list and append to the end of the list.  
**The data list must have been created by a previous "open list" command, then properly closed.**  
**Pid:** ID of the list.  
**<h..h>:** Data, in hex-ASCII, to append to the list.

DISPLAY LIST CLEAR ALL: CSI < C  
Clears the auxiliary memory list structure completely.

DISPLAY LIST CLOSE: ASCII HEX 1C (7-bit) or Control \ (8-bit)  
Close the current write-open list.

DISPLAY LIST COMPUTE/STORE CHECKSUM: CSI < a or CSI < 1a  
Computes and stores auxiliary memory checksum at first address. CSI < 1a computes and stores checksum and verifies initialization.

DISPLAY LIST EXECUTE: CSI < Pid; Py; Px x  
Open a display list for reading, and execute the commands in the list as if they were host input.  
**Pid:** ID number of the list.  
**Py:** Y offset from the origin of all graphic commands.  
**Px:** X offset from the origin of all graphic commands.

DISPLAY/DATA LIST GET INFORMATION: CSI < Pid q  
Get information about a specified display list.  
**Pid:** ID of the list.  
Response to DISPLAY/DATA LIST GET INFORMATION: CSI < Pid ; csum ;  
size ; type q  
**pid:** ID of the list.  
**csum:** Checksum for this display lists.  
**size:** List size in bytes, not including checksum byte.  
**type:** List type (0 for display, 1 for data).

DISPLAY/DATA LIST OPEN: CSI < Pid; Pt d  
Open a list for write access. Only one list may be open for writing at one time.  
**pid:** ID number of the list. Any previous list with this number is deleted. This may fragment list memory if the body directly follows the header.  
**Pt:** List type; 0=display list, 1=data list. Defaults to 0 if this parameter is missing.

DISPLAY/DATA LIST READ: CSI < Pid r  
Open a list for reading and report the data to the host.  
**pid:** ID of the list.  
Response to DISPLAY/DATA LIST READ: DCS Pid; Ps; Pt r <h..h> ST  
**pid:** ID of the list  
**Ps:** Size (in bytes) of the list.  
**Pt:** List type (0 for display, 1 for data)  
**<h..h>:** Data, in hex-ASCII, of the list.

DRAW BITMAP LIST, RELATIVE: DCS 15 G Pi ST  
DRAW BITMAP LIST, ABSOLUTE: DCS 15 G Pi; Py; Px ST

The *Draw Bitmap List* is used to draw a bit-mapped image which is stored in auxiliary memory as Bitmap List **Pi**. The image is drawn starting at the CP (relative) or from position (**Py;Px**) (absolute). By default, the image is drawn from bottom to top, to maintain consistency with the *Define Button Bitmap Label* command.

This command does not clip to the graphics clipping window.

DRAW BITMAP, RELATIVE: DCS 28 G Pw H <data> ST  
 DRAW BITMAP, RELATIVE, CORRECTED: DCS 28 G Pw; 1 H <data> ST  
 DRAW BITMAP, ABSOLUTE: DCS 28 G Py; Px; Pw H <data> ST  
 DRAW BITMAP, ABSOLUTE, CORRECTED: DCS 28 G Py; Px; Pw; 1 H <data> ST

The *Draw Bitmap* command can be used to draw a bitmap image of any size on the screen. Parameter **Pw** is the width of the bitmap in pixels. The image is drawn as data is received, rather than being buffered internally, so the maximum image size is NOT limited by buffering capacity.

By default, the image is drawn from bottom to top, to maintain consistency with the *Define Button Bitmap Label* command. If the bitmap should be drawn top to bottom, use the *Corrected* form of this command.

Note that for 16-color images, the command data contains 1 byte/pixel, while for 256-color images, the command data contains 2 bytes/pixel, MSB first.

This command does **not** clip to the graphics clipping window.

DRAW BUTTON: CSI > 3 ; Pb D  
 Draw a button on the current button page.  
**Pb**: Button number.

DRAW BUTTON PAGE: CSI > 3 D  
 Draw the current button page.

DRAW GRAPHIC OBJECT, ABSOLUTE: DCS 23 G Pid; Py; Px ST  
 DRAW GRAPHIC OBJECT, RELATIVE: DCS 23 G Pid ST  
 This command a graphic object on the screen, starting at (Py; Px) (absolute) or at the CP (relative). After the object is drawn, the new CP depends on the object type:

Object type	new CP position
3D Area	Upper-left corner of 3D area, <b>not</b> including the border.
Separator	For vertical separator, one pixel to the right of top of line. For horizontal separator, one pixel below left end of line.
Gauge	Upper-left corner of graph.
Trend Graph	Upper-left corner of graph.

DRAW KEYBOARD: CSI > 3 ; Pid K  
 Draw keyboard **Pid**.

DRAW MENU: CSI > 3 ; Pm M  
 Draws menu **Pm**.

ELLIPSE DRAW, RELATIVE: DCS 14 G Py; Px; Pb; Pa ST  
 ELLIPSE DRAW, RELATIVE: DCS 14 G Pb; Pa ST  
 Draws an ellipse on the screen centered on the reference point, which is (Py; Px) (absolute) or the CP (relative), with radii of **Pa** along X axis and **Pb** along Y axis.

ENABLE 8-BIT DATA: CSI < 3 h

ENABLE 8-BIT CONTROL: CSI < 3 l

When this function is set to **8-bit control**, the characters from 80 hex to 9F hex are treated as 8-bit control characters, and cannot be displayed literally on the screen. When it is set to **8-bit data**, these characters are treated as normal data. This function only applies to data received by the module and processed internally. The *transmit 7-bit* and *transmit 8-bit* commands determine whether the module will use 7-bit or 8-bit control codes in data which it sends back to the host.

ENTER FAST VECTOR MODE: DCS 34 G ST <fast vector data>

Enter fast vector mode of operation. Fast Vector mode is discussed elsewhere in this manual.

ENTER SELFTEST MODE: CSI ! t

Enter C9 self-test mode. This command has the same effect as installing the self-test jumper. After receipt of this command, the unit will remain in self-test until it is reset.

ERASE ALL MENUS: CSI > M

This function erases all data structures associated with ALL menus.

ERASE BUTTON BITMAP LABEL: CSI > Pb ; 1 L

Erase bitmap label from button. This command will return a command-error report if **Pb** does not exist, or if it does not have a bitmap label defined.

ERASE LINE: CSI 2 K

Erase the entire line.

ERASE LINE FROM CURSOR: CSI K or CSI 0 K

Erase from the cursor (inclusive) to the end of the line. No line attributes are affected.

ERASE LINE TO CURSOR: CSI 1 K

Erase from the beginning of the line to the cursor, inclusive. No line attributes are affected.

ERASE MENU: CSI > 1 ; Pm M

This function erases all data structures associated with menu **Pm**.

ERASE MENU ITEM: CSI > 1 ; Pm ; Pi M

Delete menu item **Pi** from menu **Pm**.

ERASE SCREEN: CSI 2 J

Erase the entire screen. The cursor maintains its original position. Screen is cleared using Graphic Objects Background color in graphics mode, and the Text Normal Background color in text mode. If the Button Autodraw flag is enabled, the current button page will be redrawn after the screen is cleared.

ERASE SCREEN FROM CURSOR: CSI J or CSI 0 J

Erase from the cursor (inclusive) to the end of the screen.

ERASE SCREEN TO CURSOR: CSI 1 J

Erase from the beginning of the screen to the cursor, inclusive. Fully erased lines are made single-height, single-width.

FLASH AREA, DEFINE: DCS 30 G *Pi*; *Pr*; *Py1*; *Px1*; *Pyu*; *Pxr*; *Pfg*; *Pbg* ST  
Define a rectangular region for flashing. The region is specified by identifying the lower-left and upper-right corner positions. This is only valid in graphics mode.

*Pi*: Area Id number, 0-15  
*Pr*: Rate: 0=4 Hz, =2 Hz, 2=1 Hz, 3=1/2 Hz  
*Py1*: Y coordinate, lower-left corner  
*Px1*: X coordinate, upper-right corner  
*Pxr*: X coordinate, upper-right corner  
*Pfg*: Foreground color of flash region (optional)  
*Pbg*: Background color of flash region (optional)

If the optional parameters *Pfg* and *Pbg* are omitted, the default Graphic Objects attribute (at the time the region is defined) will be used.

FLASH AREA, START: DCS 31 G *Pi* ST  
Start a previously defined area flashing.

*Pi*: Id of flash area, 0 - 15.

FLASH AREA, STOP: DCS 32 G *Pi* ST  
Stop a previously defined area flashing. If *Pi* is omitted, all existing flash regions will be stopped.

*Pi*: Id of flash area, 0 - 15.

GET ACTIVE BUTTON: CSI > *b*  
Request the active button on the definition button page

Response to GET ACTIVE BUTTON: CSI > *Pb* *b*  
*Pb*: button number

GET AUXILIARY MEMORY LIST REPORT: CSI > 12 *n*  
Each of these commands sends a full text report to the host, containing all pertinent information about the requested group of objects. For example, Get Button List Report returns a listing of all button pages, including all buttons on each page, showing their positions, sizes, attributes, responses and labels.

GET BAD BEAM LIST: CSI > 3 *n*  
Request report with list of bad IR beams.

Report format: CSI > *Pf* ; ... *Pf* *a*  
Report format (no bad beams): CSI > 0 *a*

GET BUTTON BITMAP STATUS: CSI > *Pb* ; 0 *L*  
Determine whether button *Pb* has a bitmap label specified.

Response to GET BUTTON BITMAP STATUS: CSI > *Pb* ; *Ps* *L*  
*Pb*: button number  
*Ps*: button bitmap status: 1=bitmap label defined, 0=no bitmap label defined.

GET BUTTON FREE SPACE: CSI > 5 *n*  
Request a report of available allocatable memory.

Report format: CSI > *free\_space* *f*

GET BUTTON LIST REPORT: CSI > 11 n  
 Each of these commands sends a full text report to the host, containing all pertinent information about the requested group of objects. For example, Get Button List Report returns a listing of all button pages, including all buttons on each page, showing their positions, sizes, attributes, responses and labels.

GET BUTTON NUMBERS: CSI > 6 n  
 Request a report of lowest (**P1**) and highest (**Ph**) button numbers used on the definition button page. On the Quick C9, **P1** is ALWAYS 0; the parameter is left there only for historical purposes.  
 Report format: CSI > P1 ; Ph b

GET BUTTON PAGE NUMBERS: CSI > P  
 Report format: CSI > Pc ; Pd P  
**Pc:** Current button page number  
**Pd:** Definition button page number

GET CHARACTER SET: CSI < S  
 Response to GET CHARACTER SET command: CSI < Ps S  
**Ps:** Character set selection (0-3).

GET COLOR ATTRIBUTES: DCS 36 G atype ST  
 Response to READ COLOR ATTRIBUTES: DCS 36 G atype; Fg;Bg ST  
**atype:** attribute type  
**Fg:** foreground color  
**Bg:** background color

GET CURRENT DRAWN KEYBOARD: CSI > 32 #  
 Request a report on which keyboard is currently drawn.  
 Response to *Get Current Drawn Keyboard*:  
 CSI > 32 1 if no keyboard is currently drawn.  
 CSI > 32; Pk h if keyboard Pk is currently drawn.

GET DEVICE STATUS: CSI 5 n  
 Host requests operating status of the terminal and/or printer. (If the terminal is in Print Control mode, the request goes to the printer, which may be unable to respond.) This command is commonly used to confirm proper communications before sending other commands.  
 Responses: CSI 0 n

GET DISPLAY LIST FREE MEMORY: CSI < 2 a  
 Request a report of available auxiliary memory.  
 Response to GET DISPLAY LIST FREE MEMORY: CSI < free\_auxmem a

GET DISPLAY INFORMATION: CSI < R  
Requests a report from the target, listing all relevant display parameters for the currently-installed flat-panel display.

Response to GET DISPLAY INFORMATION: CSI < Pt; Px; Py; Pc; Pp; Pa R  
**Pt:** Display number (0-15). This is the value of jumpers E10,E8,E7,E6  
**Px:** Highest pixel number in width. This is display width - 1.  
**Py:** Display height in pixels.  
**Pc:** Maximum colors in current video mode  
**Pp:** Points. This is the height of the standard character set in pixels.  
**Pa:** Alpha text lines. This is the number of text rows on the display.

GET FILL STYLE ATTRIBUTE: DCS 19 G ST  
Read the current value of the graphics fill pattern.  
Response to GET FILL STYLE ATTRIBUTE: DCS 19 G Ps ST

GET FIRMWARE VERSION: CSI > 9 n  
Get the current firmware version number (Pv).  
Response: CSI > Pv v

GET GRAPHIC CURSOR POSITION: DCS 1 G ST  
Read the current graphic cursor position.  
Response to GET GRAPHIC CURSOR POSITION: DCS 1 G Py ; Px ST

GET KEYBOARD LIST REPORT: CSI > 14 n  
Each of these commands sends a full text report to the host, containing all pertinent information about the requested group of objects. For example, Get Button List Report returns a listing of all button pages, including all buttons on each page, showing their positions, sizes, attributes, responses and labels.

GET LINE STYLE ATTRIBUTES: DCS 18 G ST  
Read the current values of line style and thickness.  
Response to GET LINE STYLE ATTRIBUTES: DCS 18 G Ps ; Pt ST

GET MENU LIST REPORT: CSI > 13 n  
Each of these commands sends a full text report to the host, containing all pertinent information about the requested group of objects. For example, Get Button List Report returns a listing of all button pages, including all buttons on each page, showing their positions, sizes, attributes, responses and labels.

GET NEXT BUTTON PAGE: CSI > 7 n  
Request a report of next available button page.  
Report format: CSI > Pn p

GET PRINTER STATUS: CSI ? 15 n  
To determine printer status (required before sending any print or print mode commands to the terminal).  
Responses:  
Printer ready (DTR): CSI ? 10 n  
Printer not ready (no DTR): CSI ? 11 n  
No printer: (no DTR since power-up or reset) CSI ? 13 n

GET SELFTEST ERROR COUNTER: CSI > 2 n  
 Read and clear the selftest error counter. This is NOT the same as the  
 GET SYSTEM ERROR COUNTERS command.  
 Response to GET SELFTEST ERROR COUNTER: CSI > Pe e  
     Pe: Value in selftest error counter.

GET TEXT CURSOR POSITION: CSI 6 n  
 Request cursor vertical and horizontal position.  
 Response: CSI Pv; Ph R  
     Pv: Vertical position (row, counting from 1)  
     Ph: Horizontal position (column, counting from 1)

GET TOUCH DIMENSIONS: CSI > 8 n  
 Get the touch-screen dimensions of the module (Py;Px).  
 Response: CSI > Py ; Px d

GET TOUCH POSITION: CSI > 0 n  
 Request a report from the terminal of the current touch position.  
 Report format for GET TOUCH POSITION: CSI > P1 ; Pc T  
     where P1 and Pc are the touch row and column.  
 If no touch is present, report format is CSI > T

GET VIDEO MODE: CSI v  
 Request current C9 video mode report.  
 Response to GET VIDEO MODE: CSI Pv v

GET VIDEO PAGE: CSI < P  
 Request a report of current (Pc) and definition (Pd) video page. Valid text mode video pages are  
 0-7, valid graphics mode video pages are 0-2.  
 Response to GET VIDEO PAGE command: CSI < Pc ; Pd P

GRAPH CURSOR ENABLE/DISABLE: CSI < 9 h/l  
 Enable or disable the display of the graphics cursor symbol.

GRAPH STRING CONTROL, ENABLE/DISABLE: CSI < 7 h/l  
 When enable, all ASCII control characters are interpreted as appropriate cursor control functions.  
 When off, ASCII control codes are treated as printable characters. Pertains only to the graphic  
 string draw.

GRAPH STRING WRAP, ENABLE/DISABLE: CSI < 8 h/l  
 When enabled, any attempt to draw a character outside of the graphic window will cause the  
 graphic cursor to wrap to the graphic string start position. When disabled, characters will be  
 clipped beyond the window. This function pertains only to the graphic string draw commands.

GRAPHICS CLIPPING WINDOW, ENABLE/DISABLE: CSI < 6 h/l  
 Enables (h) clipping of graphic objects at the graphic window limits. Disable (l) command is the  
 opposite.

GRAPHIC LIST CLOSE: DCS g  
 Closes a previously-opened graphic list. Only one graphic list can be opened for creation at a  
 time, and the *Graphic List Execute* command cannot be sent while a list is being created.

GRAPHIC LIST DELETE: DCS Pi d  
Deletes one existing graphic list. If **Pi** is omitted, all graphic lists are deleted.

GRAPHIC LIST EXECUTE, RELATIVE: DCS Pi x  
GRAPHIC LIST EXECUTE, ABSOLUTE: DCS Pi; Py; Px x  
Executes all graphics commands in graphic list **Pi**. The starting cursor position will be (**Py; Px**) (absolute) or the CP (relative).

GRAPHIC LIST LINK: DCS Pi; Pp g  
Associates graphic list **Pi** with button page **Pp**. Whenever button page **Pp** is drawn, graphic list **Pi** will automatically be executed. Both **Pi** and **Pp** must already exist, or an error report will be generated. If **Pi** is **-1**, any existing link on button page **Pp** will be deleted. If **Pp** is the current button page, graphic list **Pi** will be executed by this command.

GRAPHIC LIST OPEN: DCS Pi g  
Creates and opens graphic list **Pi** for writing. After this command is received, all supported graphics commands will be stored in the list and will NOT be drawn. When the desired list is completed, send the *Graphic List Close* command.

GRAPHIC LIST REPORT: CSI > 15 n  
Sends to the host a complete report of all currently-defined graphic lists, as well as their contents.

GRAPHIC STRING DRAW RELATIVE: DCS 12 G A <text string> ST  
GRAPHIC STRING DRAW ABSOLUTE: DCS 12 G Py; Px A <text string> ST  
Draws a text string on the screen, using the current attributes set by the *Set Graphics Character Attributes* and *Set Graphics String Attributes* commands. The CP is set to the lower-right corner of the string by this command.

HIGHLIGHT SCREEN TOUCH: CSI > 20 h/1  
If this option is enabled (**h**), a touch cursor will be drawn onscreen whenever the screen is touched

HORIZ TAB CLEAR: CSI g or CSI 0 g  
Clear a tab stop at the current cursor position.

HORIZ TAB CLEAR ALL: CSI 3 g  
Clear all tab stops.

HORIZ TAB SET: ESC H  
Set a tab stop at the current cursor position.

INDEX: ESC D  
Move the cursor to the line below, but keep it in the same column. If the cursor is already at the bottom of the scroll region, the screen scrolls up a line. Index is also an 8-bit control character, 84H.

KEYBOARD ENABLE/DISABLE: CSI > 33 h/1  
If this function is enabled (**h**), the build-in keyboard can be activated by touching the lower right corner of the touch screen. If this function is disabled (**1**), the keyboard cannot be activated by touch. It can still be activated using the **KEYBOARD ON** command.

**KEYBOARD ON/OFF:** **CSI > 32 h/1**  
 Activate (**h**) or deactivate (**1**) the onscreen keyboard. This command will work regardless of the state of the KEYBOARD ENABLE/DISABLE option.

**LF AFTER CR (ALPHA NEWLINE), ENABLE/DISABLE:** **CSI 20 h/1**  
 If this function is enabled (**CSI 20 h**), a linefeed (LF) appended to each carriage return (CR) (or enter key, when the numeric keypad is in numeric mode), and a CR is appended after each LF.

**LINK TREND GRAPH:** **CSI > Pid1; Pid0 o**  
 This command links trend graph **Pid1** to trend graph **Pid0**. Any number of trend graphs can be linked together. When two or more trend graphs are linked together, they are all plotted in the same space (the width and height of the first object in the list is used), and ALL of the linked graphs are updated together with one *Update Graphic Object* command.

**LOAD FONT LIST INTO CHARACTER SET:** **CSI < Pf ; Pc f**  
 Load a Font List **Pf** into one of the four user-defined character sets **Pc**. The full range of characters stored in the Font List will be placed in the specified character set; no provision is made for loading part of a font list.

**LOCK KEYBOARD:** **CSI 2 h**  
 Cause keyboard input to be ignored.

**MODULE STATE DEFAULT:** **CSI ! p**  
 Set system options back to default values. This does NOT update NVRAM.

**MODULE STATE RESTORE:** **CSI ! r**  
 Restore system options from NVRAM.

**MODULE STATE SAVE:** **CSI ! s**  
 Save system options in NVRAM.

**MULTIDROP ENABLE/DISABLE:** **CSI < Penb; Pid; Pnxtid @**  
 This command enables or disables RS485 multidrop, and defines the multidrop ID and next ID. Where C9 set-up mode is unusable, the multidrop enable command can be used to initialize the terminal to use multidrop. This initialization may be done at power up by embedding the command in display list #0 and enabling the auto-execute of list #0 power-up mode.

**Penb =**    0 disables multidrop  
               1 enables multidrop

**Pid** is the Multidrop ID for the device  
**Pnxtid** is the ID of the device to be used in the short-span termination sequence.

**MULTIDROP RECEIVE CONTROL:** **CSI Pid {**  
**Pid = 0:** All terminals receive. **Pid = 1-30:** Only terminal **Pid** receives. This is an RS485 multidrop command.

**MULTIDROP TRANSMIT CONTROL, CONTINUOUS:** **CSI Pid }**  
**Pid:** Terminal **Pid** is the designated continuous transmitter. This is an RS485 multidrop command.

MULTIDROP TRANSMIT CONTROL, SHORT-SPAN: CSI Pid |  
**pid**: Terminal Pid is the designated short-span transmitter. This is an RS485 multidrop command.

NEXT LINE: ESC E  
 Move the cursor to the start of the next line. If the cursor is already at the bottom of the scroll region, the screen scrolls up a line. Next Line is also an 8-bit control character, 85H.

NORMAL VIDEO SCREEN: CSI ? 5 l  
 Normal (not reverse) video screen.

PATTERN FILL RELATIVE: DCS 11 G ST  
 PATTERN FILL ABSOLUTE: DCS 11 G Py; Px ST  
 Fills a region of the screen beginning at the seed point, which is (**Py1**;**Px1**) (absolute) or the CP (relative), with the default fill pattern. Pattern Fill paints outward from the seed point to any boundary which is not the same color as the seed point. The CP is not changed by this command.

PERIODIC FUNCTION ENABLE: CSI < DLi; DLf; Per p  
 PERIODIC FUNCTION DISABLE: CSI < p  
 The *Periodic Function* mechanism causes the Quick C9 to automatically execute one or more display lists. The display lists will be executed sequentially from **DLi** through **DLf**, executing one display list after each **Per** seconds. Any display lists between **DLi** and **DLf** which are not present in auxiliary memory will be skipped without generating error messages. The module will continue executing the series of display lists until the *Periodic Function Disable* command is sent.

PRINT BUFFER REQUEST: CSI < 1 n  
 Host request for any buffered input from the print device.

PRINT BUFFER RESPONSE: DCS b <a..a> ST  
 Controller report of buffered input from the print device.  
**<a..a>**: Input from the print device.

PRINT CONTROL ON: CSI 5 i  
 PRINT CONTROL OFF: CSI 4 i  
 Turn Printer Control on or off. Print control mode causes host input to pass through directly to the printer without being displayed on screen. Exceptions: NUL, XON, and XOFF are not sent to the printer. Keyboard input continues to be sent to the host. Print control mode can be invoked while in auto print mode, but not vice versa.

PRINT CURSOR LINE: CSI ? 1 i  
 Prints the display line containing the cursor. The cursor position does not change. The print cursor line sequence is complete when the line prints.

PRINT FORMFEED, ENABLE/DISABLE: CSI ? 18 h/l  
 If this function is enabled, a formfeed (FF) is sent after a screen print.

PRINT SCREEN: CSI i or CSI 0 i  
 Print the entire alpha screen to the print device. There is either no print terminator character, or a form feed, depending on the state of the printer form feed mode.

PRINTER INPUT BUFFERING, ENABLE/DISABLE: CSI < 5 h/l  
Enable (**h**) or Disable (**l**) print device input buffering. If enabled, and if print input is enabled, then all input from the printer is buffered by the controller and may be requested by the host. If buffering is disabled, and print input is enabled, then all input from the print device is passed directly on to the host.

PRINTER INPUT ENABLE/DISABLE: CSI < 4 h/l  
Enable (**h**) or Disable (**l**) input recognition from the print device. If enabled, all input from the print device is passed on to the host. The data will be passed on immediately or may be buffered until requested by the host, see PRINTER INPUT BUFFER MODE below.

PRINTER PORT CONFIGURE: CSI < Pb; Pd; Ps; Pp c  
Configure the printer port serial IO.  
**Pb**: Baud rate selection (0=19200, 1=9600, 2=4800, 3=2400, 4=1200, 5=600, 6=300, 7=110).  
**Pd**: Number of data bits (0=8 bits, 1=7 bits, 2=6 bits, 3=5 bits).  
**Ps**: Number of stop bits (0=1 bit, 1=1.5 bits, 2=2 bits).  
**Pp**: Parity selection (0=none, 1=odd, 2=even).

PUSH GRAPHIC CURSOR POSITION: DCS 37 G 1 ST  
POP GRAPHIC CURSOR POSITION: DCS 37 G ST

Save/restore the current graphic cursor position. This function can be useful when executing a sequence of relative-draw commands (in display lists, graphic lists, etc.). For example, executing a *Graphic String Draw* command moves the CP a distance which may be difficult to calculate, making later relative draw commands troublesome. However, if you execute *Push Graphic Cursor Position*, draw the graphic string, then execute *Pop Graphic Cursor Position*, the CP will then be in a known location, and the length of the graphic string isn't needed.

Note that *Push/Pop Graphic Cursor Position* does NOT maintain a stack, it only stores one position; so do NOT attempt to push multiple positions. Always Push, Draw, Pop, and your drawing sequences will work predictably.

QUERY SET/RESET STATE: **CSI Pi #** or **CSI > Pi #** or **CSI < Pi #** or **CSI ? Pi #**  
where **Pi** is the desired command number.

The terminal provides a set of commands to set or clear various internal values. These commands end with either 'h' (to set an option) or 'l' (to reset an option). For example, **CSI < 19 h** enables text underline, while **CSI < 19 l** disables text underline.

A corresponding set of query commands enables the host to determine the current state of one of these values. The form of the query command is the same as the form of the related set/reset commands, except the 'h' or 'l' at the end of the command is replaced by '#'.  
**Example:**

To determine the current state of text underline, send the command CSI < 19 #  
The module will respond with one of:  
**CSI < 19 h** (if text underline is enabled), or  
**CSI < 19 l** (if text underline is disabled)

RAISE BUTTON TO TOP: CSI > Pb ; 1 b  
Raise button **Pb** above other buttons. If **Pb** is a scrolling button, this also makes it the active button.

READ GRAPHIC PIXELS: DCS 39 G c ; y ; x ST  
This command reads a series of pixels from the graphics memory area, and relays them back to the host.  
**c** = number of pixels to read  
**y** = starting scanline to read  
**x** = starting pixel column to read  
Response: DCS 40 G <DDD.....> ST

READ SYSTEM ERROR COUNTERS: CSI > E  
Request a report of the system error counters.  
Response format for READ SYSTEM ERROR COUNTERS:  
CSI > c0 ; c1 ; c2 ; c3 ; c4 ; c5 E  
If any of these counters is ever non-zero, the report should be forwarded to Deeco Engineering.

RECTANGLE DRAW RELATIVE: DCS 6 G Py2 ; Px2 ST  
RECTANGLE DRAW RELATIVE, WITH AUTOFILL: DCS 6 G Py2 ; Py1 ; Pf ST  
RECTANGLE DRAW ABSOLUTE: DCS 6 G Py1 ; Px1 ; Py2 ; Px2 ST  
RECTANGLE DRAW ABSOLUTE, WITH AUTOFILL: DCS 6 G Py1 ; Px1 ; Py2 ; Px2 ; Pf ST  
Draws a rectangle on the screen from (**Py1 ; Px1**) (absolute) or from the CP (relative) to point (**Py2 ; Px2**). If fill pattern **Pf** is provided, the rectangle is filled with the requested pattern, using the Block Draw function. The CP is set to (Y2,X2) after the operation.

REPEAT PREVIOUS TOUCH REPORT: CSI > 1 n  
Ask terminal to repeat most recent touch report. If no touch reports are enabled, or there was no previous touch, the module reports the current touch, if any. The report format for this command is the same as the report format for the most recent report, or by the GET TOUCH POSITION command.

REPORT ENTER TOUCHES, ENABLE/DISABLE: CSI > 16 h/1  
This command enables (h) or disables (l) sending of ENTER reports to the host when the screen is touched. There are four types of touch reports which the C9 can generate; ENTER, TRACKING, EXIT, MULTIPLE. Each type of report can be independently enabled/disabled.

ENTER report format: CSI > P1 ; Pc E  
where **P1** and **Pc** are the touch row and column.

REPORT EXIT TOUCHES, ENABLE/DISABLE: CSI > 17 h/1  
This command enables (h) or disables (l) sending of EXIT reports to the host when the screen is touched. There are four types of touch reports which the C9 can generate; ENTER, TRACKING, EXIT, MULTIPLE. Each type of report can be independently enabled/disabled.

EXIT report format: CSI > P1 ; Pc X  
where **P1** and **Pc** are the touch row and column.

REPORT GRAPHIC OBJECTS: CSI > 16 n  
This sends an ASCII report to the host, listing all defined graphic objects, and some information about each.

REPORT MULTIPLE TOUCHES, ENABLE/DISABLE: CSI > 19 h/1  
 This command enables (**h**) or disables (**1**) sending of MULTIPLE reports to the host when the screen is touched. A MULTIPLE touch report indicates that two or more independent touches were detected simultaneously on the screen. There are four types of touch reports which the C9 can generate; ENTER, TRACKING, EXIT, MULTIPLE. Each type of report can be independently enabled/disabled.

MULTIPLE-TOUCH report format: CSI > M

REPORT TRACKING TOUCHES, ENABLE/DISABLE: CSI > 18 h/1  
 This command enables (**h**) or disables (**1**) sending of TRACKING reports to the host when the screen is touched. There are four types of touch reports which the C9 can generate; ENTER, TRACKING, EXIT, MULTIPLE. Each type of report can be independently enabled/disabled.

TRACKING report format: CSI > P1 ; Pc T  
 where **P1** and **Pc** are the touch row and column.

REQUEST BUTTON-LIST REPORT: CSI > 11 n  
 Instruct the terminal to send back, via the serial port, a report of all buttons on all button pages.

RESET GRAPHICS CHARACTER ATTRIBUTES: DCS 16 G ST  
 Reset all Graphics Character Attributes to default values. This is equivalent to  
**DCS 16 G 0; 0; 0; 0; 0 ST.**

RESET GRAPHICS STRING ATTRIBUTES: DCS 17 G ST  
 Reset all Graphics String Attributes to default values. This is equivalent to  
**DCS 17 G 0; 0 ST.**

RESET GRAPHICS WRITE MODE: DCS 20 G ST  
 Reset all Graphics Write Mode to default value. This is equivalent to  
**DCS 20 G 0 ST.**

RESET TO INITIAL STATE: ESC c  
 Cause a hard terminal reset.

RESET TOUCH MODES: CSI > 6 D  
 Sets the value of *Report Enter/Tracking/Exit/Multiple Touch*, *Highlight Touch Mode*, *Set Button System Options*, and *Beep On Screen Touch* to default values.

REVERSE INDEX: ESC M  
 Move the cursor to the line above, but keep it in the same column. If the cursor is already at the top of the scroll region, the screen scrolls down a line. Reverse Index is also an 8-bit control character, 8DH.

REVERSE VIDEO (SCREEN): CSI ? 5 h  
 Reverse the video image of the entire screen.

SCREEN SAVER ON/OFF: CSI < 1 h/1  
 Enable/disable the screen saver time-out feature. This feature automatically blanks the screen after a specified number of minutes of inactivity (no host, keyboard, or touch input). Screen saver off disables the screen saver time-out feature.

**SCREEN SAVER TIME-OUT:** **CSI < Pn t**  
Specify the time, in minutes, of the screen saver time-out feature. This is the time of allowable inactivity before the screen is automatically blanked by the screen saver feature. The minimum time allowable is 1 minute, the maximum is 20 minutes. Default value is 10 minutes. Changing this time automatically enables the screen saver.

**SCREEN RESET ON RECEIVE, ENABLE/DISABLE:** **CSI < 15 h/1**  
This flag determines whether the screen saver is reset when data is received via the serial port.

**SCROLL UP:** **CSI Pn S**  
Cause the scroll region to scroll up.  
**Pn:** Number of lines to scroll.

**SET ACTIVE BUTTON:** **CSI > Pb b**  
Select the active button on the definition button page. This is the button which will receive all alpha text which is sent to the terminal. If Pb is not a scrolling button, the command will be rejected. This command is also called Set Active Window.  
**Pb:** button number

**SET ALPHA CHARACTER SET:** **CSI < Ps S**  
Select one of the four user-defined character set for active use. This character set is used for alpha text and graphics string draw, but NOT for button labels; button label character set is selected via the Define Button Attribute command.  
**Ps:** Character set selection (0-3).

**SET ALPHA CHARACTER SIZE:** **CSI < Ps s**  
Select the character size used for terminal characters. There are four sizes available, the basic size and 3 additional sizes. The basic size is 8x19 pixels for color displays and 8 x 16 for mono displays. The additional sizes multiply that basic size by 2, 4, and 8.  
**Ps:** Character size selection (1, 2, 3, or 4).

**SET ALPHA WRITE MODE:** **CSI < Ps M**  
Selects the raster write mode for terminal alphanumeric. The write mode may be JAM, COMPLEMENT, SET, or CLEAR. This function only works in graphics mode.  
**Pm:** Write mode (0=JAM, 1=COMPLEMENT, 2=CLEAR, 3=SET).

**SET BAD BEAM BEEP TIME:** **CSI > Pt T**  
Specify amount of time to wait, after a bad beam is detected, before a warning beep is emitted by the terminal. If Pt is 0, this warning is disabled.  
**Pt:** Delay time (in minutes). 0 = disable warning beep.

**SET BAD BEAM REPORT ENABLE/DISABLE:** **CSI > 44 h/1**  
If BAD BEAM BEEP function is enabled, enabling this option also will cause the terminal to send a warning report to the host.

Report format: **CSI > Pf ; ... Pf a**  
Report format (no bad beams): **CSI > 0 a**

**SET BUTTON PAGE:** **CSI > Pc ; Pd P**  
Select the current and definition button pages. If Pd is omitted, it is set equal to the current button page Pc.  
**Pc:** Current button page.  
**Pd:** Definition button page.

**SET BUTTON STATE:****CSI > Pb ; Ps A**

Set a multi-state button to a specific state.

**Pb:** Button number.**Ps:** Button number.**SET BUTTON SYSTEM OPTIONS:****CSI > Ps h/l**Enable (**h**) or disable (**l**) global button options.

<b>Ps</b>	<b>Button Option Name</b>
23	Button enable
24	Autodraw - draw current button page on page change
25	Select button enter ( <b>h</b> ) or exit ( <b>l</b> ) mode
26	Highlight button touches, global
27	Beep on button touch
28	Erase screen on page change

Multiple options can be enabled or disabled with one command by separating the numbers with semicolons

**SET BUTTON TIMEOUT:****CSI > Pb ; Pt B**Set the button timeout. If this value is non-zero, button **Pb** will be activated after **Pt** seconds passes with no system activity.**Pb:** Button Number.**Pt:** Button timeout, in seconds (0 = disable timeout).

SET COLOR ATTRIBUTES:

DCS 36 G atype; Fg; Bg ST

This existing command has been expanded to include default attributes for graphic objects. The new attributes are listed in the following table. Note that the original title for atype 4 has been changed.

Global Color Definitions		
atype	Foreground	Background
<b>Text</b>		
0	Color (7)	Color (0)
1	Highlighted color (0)	Highlighted Color (7)
<b>Graphic Text</b>		
2	Color (7)	Color (0)
3	Highlighted color (0)	Highlighted Color (7)
<b>Graphic Drawing</b>		
4	Graphic line, pixel, fill (7)	Screen clear & pattern fill (0)
<b>Keyboard</b>		
5	Label color (7)	Keyboard label color (0)
6	Highlighted label color (0)	Keyboard Highlighted color (7)
7	Border color (15)	Keyboard background color (0)
<b>Button</b>		
8	Normal Border Color (7)	Normal Border Color (0)
9	Highlighted Border Color (0)	Highlighted Border Color (7)
10	Normal Label Color (7)	Normal Label Background (0)
11	Highlighted Label Foreground (0)	Highlighted Label Background (7)
12	Highlighted Border (15)	Shadowed Border (8)
13	Normal Face (7)	Normal Face (0)
14	Highlighted Face (9)	Highlighted Face (0)
15	High-Water Mark (12)	Low-Water Mark (4)
16	Maximum Value Mark (11)	Minimum Value Mark (3)

16-COLOR PALETTE	
0: Black	8: Bright Black
1: Blue	9: Bright Blue
2: Green	10: Bright Green
3: Cyan	11: Bright Cyan
4: Red	12: Bright Red
5: Magenta	13: Bright Magenta
6: Brown	14: Bright Yellow
7: White	15: Bright White

**SET FILL PATTERN ATTRIBUTE:****DCS 19 G Ps ST**

Select the fill pattern for pattern fill, block draw, circle draw with autofill and rectangle fill with autodraw.

<b>Ps</b>	<b>Fill Style</b>
0	solid background
1	solid foreground
2	vertical hatch
3	horizontal hatch
4	slat left hatch
5	slat right hatch
6	sparse dot
7	wide left hatch
8	wide right hatch
9	diag cross hatch
10	horiz cross hatch
11	user-defined fill pattern

**SET GRAPHICS CHARACTER ATTRIBUTES:****DCS 16 G Ps; Pr; Pi; Pu; Pc ST**

Select the character attributes.

**Ps:** Size: 0-15 for 1-16 zoom.

**Pr:** Reverse Video Attribute: 0 for normal, 1 for reverse.

**Pi:** Italic Font Attribute: 0 for normal, 1 for slanted (italic).

**Pu:** Underline Attribute: 0 for no underline, 1 for underline.

**Pc:** Character Set (0-3).

**SET GRAPHIC CURSOR TYPE:****CSI < 10 h/1**

Selects the graphic cursor symbol. The box symbol is similar to the alpha box cursor. Its size is controlled by the graphics character size. The cross type is always a 7 X 7 cross.

**GRAPH CURSOR BOX TYPE:**

**CSI < 10 h**

**GRAPH CURSOR CROSS (+) TYPE:**

**CSI < 10 l****SET GRAPHIC CURSOR POSITION, ABSOLUTE:****DCS 1 G Py ; Px ST**

Move the graphics cursor to an absolute screen position. **Py** and **Px** are the new cursor position in pixels.

**SET GRAPHIC CURSOR POSITION, RELATIVE:****DCS 2 G Py ; Px ST**

Move the graphics cursor to a new screen position, relative to the current graphic cursor position (CP). **Py** and **Px** are the distance (in pixels) that the cursor will be moved away from the current CP, and they may be positive or negative. Note that negative parameters can be passed using a '-' sign, unlike earlier versions of this firmware.

SET GRAPHIC OBJECT MARKS: CSI > Pid; Dn; Dx; Lw; Hw; Mn; Mx p

Define/change data values for active graphic objects.

- Pid:** ID number for graphic object
- Dn:** Minimum input data value
- Dx:** Maximum input data value
- Lw:** Low-water mark
- Hw:** High-water mark
- Mn:** Enable (1) or disable (0) min-data mark
- Mx:** Enable (1) or disable (0) max-data mark

SET GRAPHICS STRING ATTRIBUTE: DCS 17 G Pd; Pp ST

Select the string attributes.

- Pd:** Direction: 0-7 for multiple of 45 degrees, counterclockwise, from right horizontal.  
This orients the characters and the entire string.
- Pp:** Path: 0-3 for left-to-right, bottom-to-top, right-to-left or top-to-bottom.

SET GRAPHIC WRITE MODE: DCS 20 G Pm ST

Select the raster write mode. The write mode refers to how new raster data (objects being drawn) and existing raster data are combined. The simplest mode is "replace" where new data completely replaces the old. There are three other methods, each described by a Boolean logical expression:

- Pm = 0** for REPLACE: NEW
- 1** for COMPLEMENT: NEW XOR OLD
- 2** for CLEAR: NEW AND OLD
- 3** for SET: NEW OR OLD:

SET LINE STYLE ATTRIBUTES: DCS 18 G Ps ; Pt ST

Select the line style and thickness for drawing graphics objects (line, circle, arc, etc.).

Ps	Line Style
0	solid foreground
1	long dash
2	short dash
3	dot
4	dot dash
5	sparse dot
6	solid background
7	user-defined line style

Pt	Line Thickness
1-20	pixels

SET RTC ATTRIBUTES, SINGLE FORM (MODIFIED):  
 SET RTC ATTRIBUTES, MULTIPLE FORM (MODIFIED):  
 Set attributes of real-time clock (RTC).

CSI < index ; value T  
 CSI < P0; ... ; Pi T

Notes on display position (index 0 and 1):  
 These coordinates are considered to be text coordinates if index 6 is set to *Alpha Text*, and are considered to be graphics coordinates if index 6 is set to *Graphics Text*. For graphics text, the row and column specify the upper-left corner of the text string. If alpha text is selected, but either row or column are too large for text coordinates (80x25), they will be converted to valid text coordinates.

Index	Description of Option
0	Display row
1	Display column
2	Foreground color
3	Background color
4	24/12-hour format
5	Time-display format (0-7) Note: this command supersedes Set RTC Display Format and makes it unnecessary, though it is still supported. 0 = no display 1 = show seconds (not valid, converts to 0) 2 = show hours 3 = show hours + seconds 4 = show date 5 = show date + seconds (not valid, converts to 4) 6 = show date + hours 7 = show date + hours + seconds
6	Alpha or Graphics text style. Note that options 7-11 apply only for graphics text style.
7	Graphics String Size
8	Graphics String Orientation (0 = horizontal, non-0 = vertical (portrait mode))
9	Underline
10	Character Set
11	Italics
12	Date Separator Character (in decimal)

SET RTC DISPLAY FORMAT:

CSI < dateOK ; hmOK ; secsOK T

This command determines which portions of the date and time are displayed. If dateOK is set, the date will be shown in MM/DD/YY format. If hmOK is set, the hours and minutes will be shown in HH:MM format, and if secsOK is also set, the hours,minutes,seconds will be shown in HH:MM:SS format. If all of these are set to zero, the time display is disabled, though the RTC will still be maintained internally.

Note that the C9 does not have a battery for maintaining time when power is off, so the date/time cannot be retained in that situation.

SET RTC TIME: CSI < Py ; Pmo ; Pd ; Ph ; Pm ; Ps T  
**Py:** Year number, in two-digit or 4-digit form  
**Pmo:** Month number (1-12)  
**Pd:** Day number (1-31)  
**Ph:** Hour number (0-23)  
**Pm:** Minute number (0-59)  
**Ps:** Second number (0-59)

SET TEXT ATTRIBUTES:  
 Select or deselect the specified attribute for all new characters received

ALL ATTRIBUTES OFF:	`CSI m or CSI 0 m
TEXT UNDERLINE ON:	CSI 4 m
TEXT UNDERLINE OFF:	CSI 24 m
TEXT BLINK ON:	CSI 5 m
TEXT BLINK OFF:	CSI 25 m
REVERSE VIDEO ON:	CSI 7 m
REVERSE VIDEO OFF:	CSI 27 m
ITALIC ON:	CSI 9m
ITALIC OFF:	CSI 29m

Multiple text attributes may be selected with one command, by separating attributes with semicolons:

Example:  
 ; turn on UNDERLINE and ITALICS with one command CSI 4 ; 9 m

**NOTE:** The Text Blink attribute can be used in graphics mode as well as text mode, though it works somewhat differently there.

SET TEXT BLINK RATE: CSI < 1 ; Pr k  
 Set the blink rate for text-mode blinking text and text cursor. This command will NOT affect graphics-mode blink rates.  
**Pr:** Blink rate (0 - 63)

SET TEXT CURSOR STYLE: CSI < 2 h/1  
 Selects the shape of the blinking alphanumeric cursor. 'h' selects the BOX style of cursor, while '1' selects the UNDERLINE style.

SET TOUCH SCREEN SENSITIVITY: CSI > Pmax ; Pmin ; Pd S  
 Set touch-screen sensitivity parameters.  
**Pmax:** Maximum-sized object that will trigger touch sensor (2-80)  
**Pmin:** Minimum-sized object that will trigger touch sensor (0-78)  
**Pd:** Delay time from initial touch to button activation, in 40msec increments.  
 Range=0-50, Default = 0.

SET USER-DEFINED FILL PATTERN: DCS 13 G P0 ; ... ; P7 ST  
 This command is used to specify the user-defined fill pattern. P0 through P7 are eight decimal parameters which represent eight eight-bit binary patterns that make up the user-defined fill pattern. This command does not set the current fill pattern to the user-defined fill pattern, it just defines the pattern.

**SET USER-DEFINED LINE STYLE:** **DCS 13 G P1 ST**  
This command is used to specify the user-defined line style. **P1** is a decimal parameter which represents a 16-bit binary pattern. This command does not set the current line style to the user-defined line style, it just defines the pattern.

**SET VIDEO MODE:** **CSI Pv v**  
Set the video mode of the terminal.  
**Pv:** video mode  
0 = text mode  
1 = VGA 16-color graphics mode  
2 = SVGA 256-color graphics mode

**SET VIDEO PAGE:** **CSI < Pc ; Pd P**  
Select current (**Pc**) and definition (**Pd**) video page. Valid text mode video pages are 0-7, valid graphics mode video pages are 0-2.

**SET-UP KEY ENABLE/DISABLE:** **CSI < 18 h/1**  
To prevent an operator from changing set-up states, it is possible to disable the set-up key on the on-screen and external keyboard. Default=enabled.

**SLEEP:** **CSI ? Pt S**  
The *sleep* command causes the main thread of the Quick C9 to pause for a short time. The time parameter **Pt** is measured in 10msec increments, with a maximum of 65535 (or almost 11 minutes). This command can be useful in display lists to leave a display on the screen for a short time before clearing it and going to the next display.

**STORE FONT LIST:** **DCS 27 G Pf ; Ps ; Pe H <data> ST**  
Stores bitmapped font data into font list **Pf**. The range of characters defined are from starting character **Ps** to ending character **Pe**. The **<data>** stream must contain 19 bytes of data for each character, regardless of the actual font size; extra bytes in each character must be zeroes in order to obtain a valid checksum. Each "byte" is sent as two-byte ASCII-HEX.

It is recommended that the MSDOS program FONTDL.EXE be used for downloading font lists to the C9.

**SYSTEM ERROR REPORTING, ACTIVE/PASSIVE:** **CSI > 35 h/1**  
The terminal has various internal error-detection mechanisms built into it. When this reporting option is passive (**1**), any detected errors are logged into a series of error counters. These counters can be read and cleared via the QUERY SYSTEM ERROR COUNTERS command. When this reporting option is active (**h**), the terminal will send a system error report, which has varying formats but always begins with **SE**. Any errors that are detected by this reporting system should be reported back to Deeco Engineering.

SYSTEM SELF-TEST:

CSI > 4 n

Initiate a system self-test. This is the same series of tests as is run at startup, except the results are reported to the host rather than displayed onscreen.

Report format:

CSI > Pf c

Pf	Checksum Error	Ram Test Error	IR Beam Failure
0	no	no	no
1	YES	no	no
2	no	YES	no
3	YES	YES	no
4	no	no	YES
5	YES	no	YES
6	no	YES	YES
7	YES	YES	YES

SYSTEM STATUS REQUEST:

CSI < 3 n

Request the current module configuration.

SYSTEM STATUS RESPONSE:

CSI < 4; Pv; Pr; Pm; Pi; Ps; Pa n

Response to the request for the system report. The response includes the firmware version and revision numbers, and the status of optional features. Each option is reported as a number ending with 0 or 1.

Pv: Firmware version number.

Pr: Firmware revision number.

Pm: unused: always 11

Pi: I-R touch option- 20 = no I-R touch detected, 21 = I-R touch detected.

Ps: Secondary serial port option- 30 = no second serial, 31 = second serial detected.

Pa: Auxiliary memory option - 40=none, 41=8K, 42=32K, 43=64K, 44=128K

TEXT UNDERLINE ENABLE/DISABLE:

CSI < 19 h/1

Determines whether alpha text is drawn with underlining.

TOGGLE SET/RESET STATE:

CSI Pi j or CSI>Pi j or CSI<Pi j or CSI ? Pi j

The terminal provides a set of commands to set or clear various internal values. These commands end with either 'h' (to set a value) or '1' (to reset a value). For example, CSI < 19 h enables text underline, while CSI < 19 1 disables text underline.

A corresponding set of toggle commands are provided to go along with the set/reset functions, and are useful when one wishes to just change a value, without knowing what its current state is.

Example:

To toggle the state of the text underline flag, send the *Toggle Text Underline* command, CSI < 19 j.

TRANSMIT 7-BIT CONTROL-CODE SEQUENCES:

ESC <space> F

TRANSMIT 8-BIT CONTROL CODES:

ESC <space> G

When the C9 sends command sequences to the host, this flag determines whether it sends escape sequences as 8-bit single-character codes, or 7-bit multiple-character sequences.

TRANSMIT 8-BIT, REQUEST STATE:

ESC <space> #

Requests the current state of the *Transmit 7/8-bit Control Sequences* flag.

UNDRAW ALL MENUS: CSI > 4 M  
This function undraws all currently-drawn menus.

UNDRAW BUTTON: CSI > 4 ; Pb D  
Undraw a button on the current button page. This does NOT delete the button, it merely undraws and deactivates it (it cannot be touched).  
Pb: Button number.

UNDRAW BUTTON PAGE: CSI > 4 D  
Undraw the current button page. This does NOT delete any of the buttons, it merely undraws and deactivates them (they cannot be touched).

UNDRAW KEYBOARD: CSI > 4 ; Pid K  
Undraw keyboard Pid.

UNDRAW MENU: CSI > 4 ; Pm M  
This function undraws menu Pm.

UNLINK TREND GRAPH: CSI > Pid o  
This command removes trend graph Pid from a set of linked trend graphs.

UNLOCK KEYBOARD: CSI 2 l  
Unlock the keyboard.

UPDATE GRAPHIC OBJECT: DCS 22 G Pid; data;... data ST  
This command updates an active graphic object with new data value(s). For gauges, there is always exactly one data value, but trend graphs may sometimes have multiple data values, if multiple trend graphs are linked together.

VECTOR DRAW, ABSOLUTE: DCS 3 G Py1 ; Px1 ; Py2 ; Px2 ST

VECTOR DRAW, RELATIVE: DCS 3 G Py ; Px ST  
Draws a vector (line) on the screen from (Py1;Px1) (absolute) or from the CP (relative) to point (Py2;Px2). The CP is set to (Y2,X2) after the operation.

VECTOR DRAW POLAR, RELATIVE: DCS 4 G Ro ; Theta ST  
 VECTOR DRAW POLAR, RELATIVE, PARTIAL: DCS 4 G Ri ; Ro ; Theta ST  
 VECTOR DRAW POLAR, ABSOLUTE: DCS 4 G Yc; Xc ; Ro ; Theta ST  
 VECTOR DRAW POLAR, ABSOLUTE, PARTIAL: DCS 4 G Yc; Xc; Ri; Ro ;Theta ST

The *Vector Draw Polar* commands are used to draw vectors using Polar coordinates, in which the length and direction of the line are specified using radius (in pixels) and an angle called Theta (in degrees). The radius and angle are measured relative to a reference point, which is the graphic cursor position (CP).

Note that there are *four* different forms of this command. The outer radius **Ro** and drawing angle **Theta** must always be specified. If the location **Yc;Xc** of the reference point is specified in the command, it is an *Absolute Vector Draw Polar*; if the reference point is NOT specified by the command, it is a *Relative Vector Draw Polar*, and the reference point is assumed to be the CP. Similarly, if ONLY radius **Ro** is specified in the command, it is a *Complete Vector Draw Polar*, and the line is drawn from the reference point out to radius **Ro**. If, however, two radius parameters **Ri** and **Ro** (Inner Radius and Outer Radius) are provided, it is a *Partial Vector Draw Polar*, and the line will only be drawn between **Ri** and **Ro**. The *Partial Vector Draw Polar* is useful for drawing such items as the sides of gauges and meters. The CP is not changed by this command.

VECTOR DRAW TO: DCS 5 G Py ; Px ST  
 This command draws a line from the CP to absolute screen position (**Py ; Px**). The CP is set to (**Py , Px**) after the operation.

WRITE GRAPHIC PIXELS: DCS 41 G y; x H <DDD.....> ST  
 This command transmits a series of ASCII-HEX digits D (valid values for D are 0-9, A-F) to the controller display. The pixels are displayed in consecutive raster-memory locations beginning at (y,x). This function is used to draw bitmapped images on the screen. It does NOT respect the graphics clipping window.

WRITE TEXT STRING: DCS 25 G Pb; FG; BG A <AAA.....> ST  
 This command transmits a text string to the active button, or to another user-selected button **Pb**.  
**Pb**: button number. If **Pb** is NOT a scrolling button, the command will be rejected. If omitted, current active button is used as target.  
**FG , BG**: Foreground and Background colors to use for text writing. If omitted, the current NORMAL\_TEXT attribute will be used.  
**<AAA...>**: string of ASCII characters.

XON/XOFF ENABLED/DISABLED: CSI < 12 h/1  
 When enabled, the terminal will send software handshaking characters to the host. If disabled, no software handshaking characters will be sent. Default=enabled.

### A.3 C9 HOST COMMAND REFERENCE, by command code

Command Code	Command Name
ASCII HEX 1C	Display List Close
CSI 1 J	Erase Screen To Cursor
CSI 1 K	Erase Line To Cursor
CSI 2 J	Erase Screen
CSI 2 K	Erase Line
CSI 2 h/l	Lock/Unlock Keyboard
CSI 20 h/l	LF After CR, Enable/Disable
CSI 3 g	Horiz Tab Clear All
CSI 4 i	Print Control Off
CSI 5 i	Print Control On
CSI ... m	Set Text Attributes
CSI 5 n	Get Device Status
CSI 6 n	Get Text Cursor Position
CSI H	Cursor Home
CSI J or CSI 0 J	Erase Screen From Cursor
CSI K or CSI 0 K	Erase Line From Cursor
CSI Pi #	Query Set/Reset Commands:
CSI Pi j	Toggle Set/Reset State
CSI Pid {	Multidrop Receive Control
CSI Pid	Multidrop Transmit Control, Short-Span
CSI Pid }	Multidrop Transmit Control, Continuous
CSI PI; Pc H	Cursor Set Position
CSI Pn A	Cursor Up
CSI Pn C	Cursor Right
CSI Pn D	Cursor Left
CSI Pn S	Scroll Up
CSI Pn T	Scroll Down
CSI g or CSI 0 g	Horiz Tab Clear
CSI i or CSI 0 i	Print Screen
CSI Pv v	Set Video Mode
CSI v	Get Video Mode
CSI ! s	Module State Save
CSI ! r	Module State Restore
CSI ! t	Enter Selftest Mode
CSI < ... @	Multidrop Enable/Disable
CSI < Pi #	Query Set/Reset State
CSI < Pi j	Toggle Set/Reset State
CSI < 1 h/l	Screen Saver On/Off
CSI < 10 h/l	Set Graph Cursor Type
CSI < 12 h/l	XON/XOFF Enabled/Disabled
CSI < 15 h/l	Screen Reset On Receive, Enable/Disable
CSI < 18 h/l	Set-Up Key Enable/Disable
CSI < 19 h/l	Text Underline Enable/Disable
CSI < 2 h/l	Set Text Cursor Style
CSI < 3 h/l	Enable 8-Bit Data/Control
CSI < 4 h/l	Printer Input Enable/Disable
CSI < 5 h/l	Printer Input Buffering, Enable/Disable
CSI < 6 h/l	Graphics Clipping Window, Enable/Disable

<b>Command Code</b>	<b>Command Name</b>
CSI < 7 h/l	Graph String Control, Enable/Disable
CSI < 8 h/l	Graph String Wrap, Enable/Disable
CSI < 9 h/l	Graph Cursor Enable/Disable
CSI < 1 n	Print Buffer Request
CSI < 2 n	Auxiliary Memory Checksum Request
CSI < 3 n	System Status Request
CSI < Pa A	Define Auto-execute Mode
CSI < C	Display List Clear All
CSI < Pl; Pi C	Delete Auxiliary Memory List
CSI < Ps M	Set Alpha Write Mode
CSI < P	Get Video Page
CSI < Pc ; Pd P	Set Video Page
CSI < R	Get Display Information
CSI < S	Get Character Set
CSI < Ps S	Set Character Set
CSI < . . . T	Set RTC Attributes
CSI < a or CSI < 1a	Display List Compute/Store Checksum
CSI < 2 a	Get Display List Free Memory
CSI < . . . c	Printer Port Configure
CSI < Pid; Pt d	Display/Data List Open
CSI < e	Clear Scrolling Window
CSI < Pf ; Pc f	Load Font List Into Character Set
CSI < 0; Pr k	Enable/Disable Text Blink
CSI < 1 ; Pr k	Set Text Blink Rate
CSI < . . . p	Periodic Function Enable/Disable
CSI < Pid q	Display/Data Get Information
CSI < Pid r	Display/Data List Read
CSI < Ps s	Set Character Size
CSI < Pn t	Screen Saver Time-Out
CSI < . . . w	Define Graphic Clipping Window
CSI < . . . x	Display List Execute
CSI > Pb ; 1 b	Raise Button To Top
CSI > b	Get Active Button
CSI > Pb b	Set Active Button
CSI > Ps h/l	Set Button System Options
CSI > 16 h/l	Report Enter Touches, Enable/Disable
CSI > 17 h/l	Report Exit Touches, Enable/Disable
CSI > 18 h/l	Report Tracking Touches, Enable/Disable
CSI > 19 h/l	Report Multiple Touches, Enable/Disable
CSI > 20 h/l	Highlight Screen Touch
CSI > 21 h/l	Beep On Screen Touch
CSI > 32 h/l	Keyboard On/Off
CSI > 33 h/l	Keyboard Enable/Disable
CSI > 34 h/l	Command Error Reporting, Enable/Disable
CSI > 35 h/l	System Error Reporting, Active/Passive
CSI > 44 h/l	Set Bad Beam Report Enable/Disable
CSI > Pi j	Toggle Set/Reset State
CSI > 0 n	Get Touch Position
CSI > 1 n	Repeat Previous Touch Report
CSI > 2 n	Get Selftest Error Counter
CSI > 3 n	Get Bad Beam List

Command Code	Command Name
CSI > 4 n	System Self-Test
CSI > 5 n	Query Button Free Space
CSI > 6 n	Query Button Numbers
CSI > 7 n	Query Next Button Page
CSI > 8 n	Get Touch Dimensions
CSI > 9 n	Get Firmware Version
CSI > 11 n	Get Button List Report
CSI > 12 n	Get Auxiliary Memory List Report
CSI > 13 n	Get Menu List Report
CSI > 14 n	Get Keyboard List Report
CSI > 15 n	Graphic List Report
CSI > 16 n	Report Graphic Objects
CSI > Pid o	Unlink Trend Graph
CSI > Pid1; Pid0 o	Link Trend Graph
CSI > Pid; . . . p	Set Graphic Object Marks
CSI > . . . t	Define Text Button Area
CSI > Pi #	Query Set/Reset State
CSI > Pb A	Activate Button
CSI > Pb ; Ps A	Set Button State
CSI > B	Delete All Buttons
CSI > . . . B	Define Button Area
CSI > Pb B	Delete Button (All Button Pages)
CSI > Pb ; Pt B	Set Button Timeout
CSI > Pb C	Delete Button (Definition Button Page Only)
CSI > . . . C	Copy Button Responses
CSI > 1 D	Delete All Buttons
CSI > 2 D	Delete Button Page
CSI > 3 ; Pb D	Draw Button
CSI > 3 D	Draw Button Page
CSI > 4 ; Pb D	Undraw Button
CSI > 4 D	Undraw Button Page
CSI > 6 D	Reset Touch Modes
CSI > E	Read System Error Counters
CSI > . . . F	Define Button Attributes
CSI > . . . G	Define Button System Attribute
CSI > K	Delete All Keyboards
CSI > 1 ; Pid K	Delete Keyboard
CSI > 3 ; Pid K	Draw Keyboard
CSI > 4 ; Pid K	Undraw Keyboard
CSI > Pid ; -1 ; . . . K	Define Keyboard Attributes, Single/ Multiple Form
CSI > Pid ; Kid ; . . . K	Define Key Attributes, Single/Multiple Form
CSI > Pb ; 0 L	Get Button Bitmap Status
CSI > Pb ; 1 L	Delete Button Bitmap Label
CSI > M	Erase All Menus
CSI > Pm ; -1 ; . . . M	Define Menu Attribute, Single/ Multiple Form
CSI > Pm ; Pi ; . . . M	Define Menu Item Attributes
CSI > 1 ; Pm M	Erase Menu
CSI > 1 ; Pm ; Pi M	Erase Menu Item
CSI > 3 ; Pm M	Draw Menu
CSI > 4 M	Undraw All Menus

Command Code	Command Name
CSI > 4 ; Pm M	Undraw Menu
CSI > Pid O	Delete Graphic Object
CSI > Pid; . . . value O	Define Graphic Object Attributes, Single/Multiple Form
CSI > P	Get Button Page Numbers
CSI > ... P	Set Button Page
CSI > ... S	Set Touch Screen Sensitivity
CSI > ... T	Set Bad Beam Beep Time
CSI > 32 #	Get Current Drawn Keyboard
CSI ? Pi #	Query Set/Reset State
CSI ? Pi j	Toggle Set/Reset State
CSI ? 1 i	Print Cursor Line
CSI ? 15 n	Get Printer Status
CSI ? 8 h/l	Auto Repeat On/Off
CSI ? 18 h/l	Print Formfeed, Enable/Disable
CSI ? 25 h/l	Text Cursor On/Off
CSI ? 4 i	Auto Print Off
CSI ? 5 h	Reverse Video (Screen)
CSI ? 5 i	Auto Print On
CSI ? 5 l	Normal Video (Screen)
CSI ? Pt S	Sleep
DCS 0 G ST	Clear Graphics Window
DCS 1 G ST	Get Graphic Cursor Position
DCS 1 G . . . ST	Set Graphic Cursor Position, Absolute
DCS 2 G . . . ST	Set Graphic Cursor Position, Relative
DCS 3 G ... ST	Vector Draw, Absolute/Relative
DCS 4 G . . . ST	Vector Draw Polar
DCS 5 G . . . ST	Vector Draw To
DCS 6 G . . . ST	Rectangle Draw
DCS 7 G . . . ST	Circle Draw
DCS 8 G . . . ST	Arc Draw
DCS 9 G ... ST	Block Fill Draw
DCS 11 G . . . ST	Pattern Fill
DCS 12 G . . . A <text string> ST	Graphic String Draw
DCS 13 G PI ST	Set User-Defined Line Style
DCS 13 G P0 ; ... ; P7 ST	Set User-Defined Fill Pattern
DCS 14 G . . . ST	Ellipse Draw, Relative/Absolute
DCS 15 G . . . ST	Draw Bitmap Relative/Absolute
DCS 16 G ... ST	Set Graphics Character Attributes
DCS 16 G ST	Reset Graphics Character Attributes
DCS 17 G ST	Reset Graphics String Attributes
DCS 17 G ... ST	Set Graphics String Attribute
DCS 18 G ST	Get Line Style Attributes
DCS 18 G Ps ; Pt ST	Set Line Style Attributes
DCS 19 G ST	Get Fill Style Attribute
DCS 19 G Ps ST	Set Fill Style Attribute
DCS 20 G Pm ST	Set Graphic Write Mode
DCS 20 G ST	Reset Graphics Write Mode
DCS 21 G Pid; 0; . . . ST	Define 3D Area (type=0)
DCS 21 G Pid; 1 . . . ST	Define 3D Separator (type=1)

Command Code	Command Name
DCS 21 G Pid; 2 . . . ST	Define Gauge (type=2)
DCS 21 G Pid; 3 . . . ST	Define Trend Graph (type=3)
DCS 22 G Pid ST	Clear Trend Graph
DCS 22 G Pid; data; . . . data ST	Update Graphic Object
DCS 23 G Pid; . . . ST	Draw Graphic Object, Absolute/Relative
DCS 24 G . . . ST	Bitmap List Tile
DCS 25 G . . . A <AAA.....> ST	Write Text String
DCS 26 G . . . H <ddd_> ST	Define Button Bitmap Label
DCS 27 G . . . H <data> ST	Store Font List
DCS 28 G . . . <data> ST	Draw Bitmap
DCS 29 G . . . H <data> ST	Define Bitmap List
DCS 30 G . . . ST	Flash Area, Define
DCS 31 G . . . ST	Flash Area, Start
DCS 32 G . . . ST	Flash Area, Stop
DCS 33 G . . . ST	Define Character Bitmap
DCS 34 G ST <fast vector data>	Enter Fast Vector Mode
DCS 35 G . . . ST	Bitmap List Fill, Absolute/Relative
DCS 36 G atype; Fg; Bg ST	Set Color Attributes
DCS 36 G atype ST	Get Color Attributes
DCS 37 G ST	Pop Graphic Cursor Position
DCS 37 G 1 ST	Push Graphic Cursor Position
DCS 39 G c; y; x ST	Read Graphic Pixels
DCS 41 G y; x H <DDD.....> ST	Write Graphic Pixels
DCS g	Graphic List Close
DCS Pi d	Graphic List Delete
DCS Pi g	Graphic List Open
DCS Pi; Pp g	Graphic List Link
DCS Pid a <h...h> ST	Display/Data List Append
DCS b <a..a> ST	Print Buffer Response
DCS . . . x	Graphic List Executive, Relative/Absolute
DCS ~ Pb / Rh : Rl : Rb : Rg ST	Define Button Responses
DCS % Pm ; Py ; Px / host   local   label % Pnext / ... ST	Define Menu Responses
DCS ? Pid ; Py ; Px & responses % width / responses % width ... ST	Define Keyboard Responses
DCS& Pid / . . . ST	Define Control Responses
ESC <space> #	Transmit 8-bit, request state
ESC <space> F	Transmit 7-Bit Control-Code Sequences.
ESC <space> G	Transmit 8-Bit Control Codes
ESC c	Reset To Initial State
ESC D	Index
ESC E	Next Line
ESC H	Horiz Tab Set
ESC M	Reverse Index



## APPENDIX B: Command Error Reporting

The C9 supports a command for generating error reports when invalid commands are received from the host. These reports, although they are made cryptic to reduce serial-port overhead, provide detailed information about format and data errors which are detected in host commands. The *Command-Error Reporting Mode* is an excellent tool for debugging user programs which are not functioning properly.

### B.1 The Error Reporting Mode command

C9 Error Reporting is enabled via the *Enable Command-Error Reporting* command, `CSI>34h`. It is disabled by sending `CSI>341`, and is disabled by default. When error reporting is enabled, any detectable error in the format of C9 commands will generate a report back to the host. The format of the error report varies, depending upon the type of error, and is discussed in the following section.

### B.2 Error Report Formats

This section describes the different types of errors which can be detected and reported by the C9. It will show the report format for each error, and give an example of an erroneous command and the report which would result.

Error Code	Description of Error Report
1	Invalid button coordinate Report format: <code>ER1_ButtonNumber</code> <u>Example:</u> Command Sent: <code>CSI &gt; 1 ; 1 ; 1 ; 25 ; 100 B</code> Report Returned: <code>ER1_1</code> Meaning: "I received a <i>Define Button Area</i> command for button number 1, but it contained an invalid button coordinate."
2,3	Out Of Memory (cannot store new data)
4	Invalid argument count in command Report format: <code>ER4_Cgroup_ArgCount_Command</code> <u>Example:</u> Command Sent: <code>CSI &gt; 2 ; 1 n</code> Report Returned: <code>ER4_27_2_n</code>  <b>Cgroup</b> is the command group, or type of command, which exhibited the error. These values are listed later in this appendix. In the above example, Cgroup is 27, which means it was a <code>CSI &gt;</code> command.  <b>ArgCount</b> is the number of arguments received with the command, which was considered to be invalid.  <b>Command</b> is the letter which terminated and defined the command. In this example that was <code>n</code> .  Meaning: "I received a <code>CSI &gt; ... n</code> command with two arguments, which is not valid for that command."

Error Code	Description of Error Report
5	Invalid index in command Report format: <b>ER5_Cgroup_Index_Command</b> <u>Example:</u> Command Sent: <b>CSI ? 3 h</b> Report Returned: <b>ER5_7_3_h</b> Meaning: "I received a <b>CSI ? 3 h</b> command, and 3 is not a valid index for that command."
6	Requested button number does not exist on appropriate button page Report format: <b>ER6_Cgroup_ButtonNumber_Command</b> <u>Example:</u> Command Sent: <b>CSI &gt; 5 A</b> Report Returned: <b>ER6_27_5_A</b> Meaning: "I received a <b>CSI &gt; 5 A</b> command, but button number 5 does not exist, so I cannot perform that command on it."
7	Invalid page number in command Report format: <b>ER7_Cgroup_Page_Command</b> <u>Example:</u> Command Sent: <b>CSI &gt; 130 P</b> Report Returned: <b>ER7_27_130_P</b>  Meaning: "I received a <b>CSI &gt; 130 P</b> command, but page number 130 is not valid, so I cannot perform the requested command."
8	Attempt to draw a menu with no items in it Report format: <b>ER8</b>
9	Unsupported command received Report format: <b>ER9_Cgroup_Command</b> <u>Example:</u> Command Sent: <b>CSI &lt; B</b> Report Returned: <b>ER9_9_B</b> Meaning: "I received a <b>CSI &lt; B</b> command, which is not supported in the C9."
10	Wrong Video Mode: received a DCS command which is only valid in graphics mode Report format: <b>ER10_Index</b> <u>Example:</u> Command Sent: <b>DCS 5 G 50 ST</b> Report Returned: <b>ER10_5</b> Meaning: "I received a <b>DCS 5 G ... ST</b> command, which is not supported in text mode.  Note: The only DCS commands which are supported in text mode are <b>25, 26, 27, 33, 36, and 37.</b>

Error Code	Description of Error Report
11	<p>Keyboard active: video mode and current video page cannot be changed while an onscreen keyboard is active. Also draw/undraw keyboard. Report format: <b>ER11_PageReq_Command</b></p>
12	<p>Bad Command Format in a <i>Define Button Response</i> command. Report format: <b>ER12_ButtonNumber</b></p>
13	<p>Non-existent Button State specified in <i>Set Button State</i> command. Report format: <b>ER13_ButtonNumber_ButtonState</b></p>
14	<p>An invalid attribute-set command was sent for button 0. Button 0 is a dedicated button that is ALWAYS full-screen, non-bordered and scrolling. Any Define Button Attribute command which attempts to change these attributes on button 0 will generate this error report. Report format: <b>ER14_Cgroup_Command</b></p>
15	<p>Invalid button operation. A button command was received which is not supported for some reason. For example, sending Delete Button Bitmap Label to a button which does not have a bitmap label defined will generate this error. Report format: <b>ER15_ButtonNumber_Command</b></p>
16	<p>Invalid argument count in <i>DCS ... G</i> (graphics) command Report format: <b>ER16_ArgCount_nnG</b> <u>Example:</u> Command Sent: <b>DCS 9 G 100 ; 200 ; 100 ST</b> Report Returned: <b>ER16_3_9G</b> Meaning: "I received a <i>DCS 9 G</i> command with 3 arguments, which is not valid for that command."</p>
17	<p>Invalid argument value in <i>DCS ... G</i> (graphics) command Report format: <b>ER17_ArgValue_nnG</b> <u>Example:</u> Command Sent: <b>DCS 9 G 100 ; 800 ST</b> Report Returned: <b>ER17_800_9G</b> Meaning: "I received a <i>DCS 9 G</i> command with an argument value of 800, which is not valid for that command."</p>
18	<p>Item does not exist: a command was received which was applied to an object which is not defined. Report format: <b>ER18_Cgroup_ItemID_command</b> <u>Example:</u> Command Sent: <b>CSI &gt; 3 ; 1 M</b> Report Returned: <b>ER18_27_1 M</b> Meaning: "I received a <i>CSI &gt; ... M</i> command affecting object number 1, (in this case, Menu number 1), which does not currently exist."</p>

Error Code	Description of Error Report
19	<p>No Auxiliary Memory Present.  Report format: <b>ER19_command</b>  <u>Example:</u>  Command Sent: <b>CSI &lt; 2 x</b>  Report Returned: <b>ER19_x</b>  Meaning: "I received a command which needs to access auxiliary memory, but there is none installed."</p>
20	<p>Bad auxiliary memory list checksum.  Report format: <b>ER20_Cgroup_ItemID_command</b>  <u>Example:</u>  Command Sent: <b>CSI &lt; 1 x</b>  Report Returned: <b>ER20_9_1_x</b>  Meaning: "I received a <b>CSI &lt; 1 x</b> command, but display list 1 has a bad checksum so it cannot be executed."</p>
21	<p>Invalid auxiliary memory list type: a command was applied to the wrong auxmem list type.  Report format: <b>ER21_Cgroup_ItemID_command</b>  <u>Example:</u>  Command Sent: <b>CSI &lt; 1 x</b>  Report Returned: <b>ER21_9_1_x</b>  Meaning: "I received a <b>CSI &lt; 1 x</b> command, but auxmem list 1 is not a display list."</p>
22	<p>Invalid keyboard command: Attempt to re-define one of the system keyboards. Keyboards 0, 1, 2 are system keyboards and cannot be redefined.  Report format: <b>ER22_kbdID_command</b></p>
23	<p>Invalid value in command.  Report format: <b>ER23_Cgroup_value_command</b>  <u>Example:</u>  Command Sent: <b>CSI &gt; 3; -1; 0; 4 K</b>  Report Returned: <b>ER23_27_4_K</b>  Meaning: "I received a <b>CSI &gt; ... K</b> with an argument value of 4, which is not valid for that command."</p>
24	<p>Invalid Display List chain in auxiliary memory.  The display list chain has become corrupted; send the <i>Display List Clear All</i> command to restore auxiliary memory consistency.</p>
25	<p>Invalid data count in Store Font List command. The String Terminator code was detected before the expected number of bytes was received.  Report format: <b>ER25</b></p>
26	<p>Item in use. An attempt was made to execute or delete a graphic list while another graphic list was currently being defined. Close the current graphic list before running any other graphic list commands.  Report format: <b>ER26_Cgroup_ItemID_command</b></p>
27	<p>Auxiliary Memory Overrun.  Attempt to write past end of device. (i.e., device is full)  Report format: <b>ER27_</b></p>

## Command groups (Cgroups)

A **Cgroup** is a numeric code which represents a class of commands, such as **CSI**, **DCS**, etc. They are used by the error-reporting mechanism to generate informative error reports. This table lists all C9 command groups which may generate error reports. All unlisted group numbers are internal states which do not generate error messages.

Command Group	Description
1	ESC
2	ESC Space
3	ESC P
5	ESC P ... ESC
6	CSI
7	CSI ?
8	CSI !
9	CSI <
11	ESC P ... G ... ESC
27	CSI >
28	DCS ~
29	DCS ~ ... /



## APPENDIX C: ASCII Characters And Equivalents

<u>Decimal</u>	<u>Hexadecimal</u>	<u>Control Char</u>	<u>Command</u>
0	00H	NUL	^@
1	01H	SOH	^A
2	02H	STX	^B
3	03H	ETX	^C
4	04H	EOT	^D
5	05H	ENQ	^E
6	06H	ACK	^F
7	07H	BEL	^G
8	08H	BS	^H
9	09H	HT	^I
10	0AH	LF	^J
11	0BH	VT	^K
12	0CH	FF	^L
13	0DH	CR	^M
14	0EH	SO	^N
15	0FH	SI	^O
16	10H	DLE	^P
17	11H	DC1	^Q
18	12H	DC2	^R
19	13H	DC3	^S
20	14H	DC4	^T
21	15H	NAK	^U
22	16H	SYN	^V
23	17H	ETB	^W
24	18H	CAN	^X
25	19H	EM	^Y
26	1AH	SUB	^Z
27	1BH	ESC	^[
28	1CH	FS	^\
29	1DH	GS	^]
30	1EH	RS	^^
31	1FH	US	^_

Dec	Hex	Chr	Dec	Hex	Chr	Dec	Hex	Chr
32	20H		62	3EH	>	92	5CH	\
33	21H	!	63	3FH	?	93	5DH	]
34	22H	"	64	40H	@	94	5EH	^
35	23H	#	65	41H	A	95	5FH	~
36	24H	\$	66	42H	B	96	60H	¯
37	25H	%	67	43H	C	97	61H	a
38	26H	&	68	44H	D	98	62H	b
39	27H	'	69	45H	E	99	63H	c
40	28H	(	70	46H	F	100	64H	d
41	29H	)	71	47H	G	101	65H	e
42	2AH	*	72	48H	H	102	66H	f
43	2BH	+	73	49H	I	103	67H	g
44	2CH	,	74	4AH	J	104	68H	h
45	2DH	-	75	4BH	K	105	69H	i
46	2EH	.	76	4CH	L	106	6AH	j
47	2FH	/	77	4DH	M	107	6BH	k
48	30H	0	78	4EH	N	108	6CH	l
49	31H	1	79	4FH	O	109	6DH	m
50	32H	2	80	50H	P	110	6EH	n
51	33H	3	81	51H	Q	111	6FH	o
52	34H	4	82	52H	R	112	70H	p
53	35H	5	83	53H	S	113	71H	q
54	36H	6	84	54H	T	114	72H	r
55	37H	7	85	55H	U	115	73H	s
56	38H	8	86	56H	V	116	74H	t
57	39H	9	87	57H	W	117	75H	u
58	3AH	:	88	58H	X	118	76H	v
59	3BH	;	89	59H	Y	119	77H	w
60	3CH	<	90	5AH	Z	120	78H	x
61	3DH	=	91	5BH	[	121	79H	y

Dec	Hex	Chr	Dec	Hex	Chr	Dec	Hex	Chr
122	7AH	z	152	98H	ÿ	182	B6H	⌌
123	7BH	{	153	99H	Ö	183	B7H	⌍
124	7CH		154	9AH	Ü	184	B8H	⌎
125	7DH	}	155	9BH	ç	185	B9H	⌏
126	7EH	~	156	9CH	£	186	BAH	⌐
127	7FH	□	157	9DH	¥	187	BBH	⌑
128	80H	Ç	158	9EH	₤	188	BCH	⌒
129	81H	ü	159	9FH	f	189	BDH	⌓
130	82H	é	160	A0H		190	BEH	⌔
131	83H	â	161	A1H	í	191	BFH	⌕
132	84H	ä	162	A2H	ó	192	C0H	⌖
133	85H	à	163	A3H	ú	193	C1H	⌗
134	86H	â	164	A4H	ñ	194	C2H	⌘
135	87H	ç	165	A5H	Ñ	195	C3H	⌙
136	88H	è	166	A6H	ª	196	C4H	⌚
137	89H	ë	167	A7H	º	197	C5H	⌛
138	8AH	è	168	A8H	¿	198	C6H	⌜
139	8BH	ï	169	A9H	⌈	199	C7H	⌝
140	8CH	î	170	AAH	⌋	200	C8H	⌞
141	8DH	ì	171	ABH	½	201	C9H	⌟
142	8EH	Ä	172	ACH	¼	202	CAH	⌠
143	8FH	Å	173	ADH	;	203	CBH	⌡
144	90H	É	174	AEH	«	204	CCH	⌢
145	91H	æ	175	AFH	»	205	CDH	⌣
146	92H	Æ	176	B0H	⦿	206	CEH	⌤
147	93H	ô	177	B1H	⦿	207	CFH	⌥
148	94H	ö	178	B2H	⦿	208	D0H	⌦
149	95H	ò	179	B3H	⦿	209	D1H	⌧
150	96H	ù	180	B4H	⦿	210	D2H	⌨
151	97H	ù	181	B5H	⦿	211	D3H	〈

Dec	Hex	Chr	Dec	Hex	Chr
212	D4H	↳	242	F2H	≥
213	D5H	ƒ	243	F3H	≤
214	D6H	ƒ	244	F4H	∫
215	D7H	ƒ	245	F5H	∫
216	D8H	ƒ	246	F6H	÷
217	D9H	ƒ	247	F7H	≈
218	DAH	ƒ	248	F8H	°
219	DBH	■	249	F9H	•
220	DCH	■	250	FAH	·
221	DDH	■	251	FBH	√
222	DEH	■	252	FCH	n
223	DFH	■	253	FDH	2
224	E0H	α	254	FEH	■
225	E1H	β	255	FFH	□
226	E2H	Γ			
227	E3H	π			
228	E4H	Σ			
229	E5H	σ			
230	E6H	μ			
231	E7H	τ			
232	E8H	Φ			
233	E9H	Θ			
234	EAH	Ω			
235	EBH	ð			
236	ECH	∞			
237	EDH	∅			
238	EEH	ε			
239	EFH	∩			
240	F0H	≡			
241	F1H	±			

# Index

## —3—

3D area, 84  
3D separator, 84

## —A—

activate button, 81  
active button, 96, 106  
active window, 106  
all attributes off, 112  
alpha character size, 93, 96  
alpha cursor, 81  
alpha newline, 81  
alpha write mode, 106  
alternate character set, 56, 58, 62, 106  
ANSI, 3, 4, 5  
ANSI 3.64, 1, 2, 3  
arc draw, 47, 49, 81  
area flash, 96  
ASCII, 3, 4, 5, 28, 33, 43, 44, 52, 54, 55, 56, 70, 71, 92, 99, 116, 129  
auto draw, 86  
auto print, 81, 102  
auto repeat, 81  
auto-configure, 71  
auto-execute mode, 84  
autofill, 47, 109  
auxiliary memory, 54, 71, 81, 92, 93, 114  
auxiliary memory checksum request, 81  
auxiliary memory checksum response, 81  
auxiliary memory list, 92  
auxiliary memory list report, 96

## —B—

bad beam beep time, 106  
bad beam list, 96  
bad beam report, 106  
bad-beam notification, 45  
baud rate, 103  
beep, 43, 45, 107  
beep on button touch, 86  
beep on screen touch, 82  
beep on touch report, 86  
bitmap, 23, 27, 56, 57, 95, 96, 125  
bitmap list, 84  
bitmap list fill, 82  
bitmap list time, 82

block draw, 47, 49, 82, 104, 109  
buffered-style keyboard, 21, 40  
button bitmap label, 83, 86  
button bitmap labels, 27  
button bitmap status, 96  
button border width for graphics buttons, 86  
button commands, 22, 28, 30  
button enable, 86  
button enter/exit mode, 86  
button free space, 96  
button labels, 106  
button list report, 97  
button location, 123  
button numbers, 97  
button page numbers, 97  
button pages, 30, 35, 83, 91, 92, 105, 106  
button response, 7, 23, 26, 27, 29, 30, 35, 83, 86, 125  
button state, 107  
button structure, 22  
button system options, 107  
button timeout, 107  
button zero, 29, 33, 35  
buttons, 1, 21, 22, 25, 28, 29, 30, 31, 32, 33, 34, 35, 39, 40, 71, 86, 91, 92, 96, 97, 98, 104, 105, 115  
buttons on the fly, 32

## —C—

character attribute select, 55  
character attributes, 3, 109  
character set, 97, 101, 106  
character set select, 55, 97, 99, 106, 107  
character size, 106  
circle draw, 47, 49, 82, 109  
clear active button, 35  
clear all lists, 92  
clear graphics clipping window, 82  
clear scrolling window, 82  
clear trend graph, 82  
close list, 92  
color attributes, 27, 97  
command description, 3, 5  
command error reporting, 82  
command groups, 127  
command reference, 73  
command string initiator, 5  
compute/store checksum, 93  
configuration inquiry, 70  
control responses, 87  
copy button bitmap label, 83

- copy button command, 34
- copy button response, 34, 83
- correct corrupted auxmem lists, 83
- corrupted auxmem lists, 83
- current drawn keyboard, 97
- current page, 30
- cursor down, 83
- cursor home, 83
- cursor left, 83
- cursor position, 3, 4, 54, 83, 100, 102
- cursor right, 83
- cursor up, 83

## —D—

- data/display lists, 93
- default settings, 43
- define 3D area, 84
- define 3D separator, 84
- define alternate character, 55, 57
- define auto-execute mode, 84
- define bitmap list, 84
- define button area, 31, 84
- define button attribute, 34, 85
- define button bitmap label, 86
- define button response, 31, 34, 86
- define button system attribute, 86
- define button text area, 29
- define character bitmap, 87
- define control responses, 87
- define gauge, 87
- define graphic object attributes, 87
- define graphics clipping window, 87
- define key attributes, 89
- define keyboard attributes, 40, 89
- define keyboard responses, 40, 90
- define menu attribute, 90
- define menu item attributes, 90
- define menu responses, 37, 91
- define text button area, 26, 29, 33, 91
- define trend graph, 91
- definition page, 30
- delete all buttons, 91
- delete all keyboards, 92
- delete auxiliary memory list, 92
- delete button page, 92
- delete graphic object, 92
- delete keyboard, 92
- device command strings, 5
- device status, 97
- display, 1, 3, 4, 6, 22, 23, 25, 30, 33, 34, 35, 54, 56, 57, 58, 71, 72, 92, 93, 99, 101, 102, 116
- display information, 98
- display list clear all, 92
- display list close, 92
- display list computer/store checksum, 93
- display list execute, 93
- display list free memory, 97
- display lists, 54

- display/data list append, 92
- display/data list get information, 93
- display/data list open, 93
- display/data list read, 93
- draw bitmap, 94
- draw bitmap list, 93
- draw button, 94
- draw button page, 94
- draw graphic object, 94
- draw keyboard, 40, 41, 94
- draw menu, 37, 94

## —E—

- ellipse draw, 94
- enable 8-bit control, 95
- enable 8-bit data, 95
- enter fast vector mode, 54, 95
- enter selftest mode, 95
- entry mode, 43
- erase all menus, 95
- erase button bitmap label, 95
- erase line, 95
- erase line from cursor, 95
- erase line to cursor, 95
- erase menu, 95
- erase menu item, 95
- erase screen, 35, 95
- erase screen from cursor, 95
- erase screen on page change, 107
- erase screen to cursor, 95
- erasing objects, 51
- error, 6, 7, 23, 43, 44, 70, 82, 95, 99, 104, 113, 114, 123, 124, 125, 126, 127
- error codes, 6
- escape sequences, 4, 114
- execute display list, 71, 72, 93
- exit mode, 43

## —F—

- fast vector, 52, 53, 54, 95
- fast vector mode, 52, 53, 54, 95
- fill pattern, 49
- fill pattern attribute, 109
- fill style attribute, 98
- firmware version, 98
- flash area, 96
- font list, 55, 56, 113
- fonts, 55, 56
- free space, 96

## —G—

- gauge, 87
- get active button, 96
- get auxiliary memory list report, 71, 96
- get bad beam list, 96
- get button bitmap status, 96

get button free space, 96  
 get button list report, 71, 97  
 get button numbers, 97  
 get button page numbers, 97  
 get character set, 97  
 get color attributes, 97  
 get current drawn keyboard, 97  
 get device status, 97  
 get display information, 98  
 get display list free memory, 97  
 get fill style attribute, 98  
 get firmware version, 98  
 get graphic cursor position, 98  
 get keyboard list report, 98  
 get line style attributes, 98  
 get menu list report, 98  
 get next button page, 98  
 get printer status, 98  
 get selftest error counter, 99  
 get text cursor position, 99  
 get touch dimensions, 99  
 get touch position, 99  
 get video mode, 99  
 get video page, 99  
 graph cursor, 99  
 graph cursor type, 109  
 graph string control, 99  
 graph string wrap, 99  
 graphic cursor position, 98, 109  
 graphic list, 99, 100  
 graphic list delete, 100  
 graphic list execute, 100  
 graphic list link, 100  
 graphic list open, 100  
 graphic object, 2, 3, 92, 95, 96, 99  
 graphic object attributes, 87  
 graphic object marks, 110  
 graphic pixels, 104, 116  
 graphic string draw, 100  
 graphic terminal operation, 2  
 graphic text, 51, 108  
 graphic window, 50, 51, 87, 99  
 graphic write mode, 110  
 graphics clipping, 50  
 graphics clipping window, 99  
 graphics mode, 33  
 graphics string attribute, 110

## —H—

highlight button touches, 107  
 highlight button when touched, 86  
 highlight graphics buttons, 86  
 highlight screen touch, 100  
 highlight touch mode, 43  
 horiz tab clear, 100  
 horiz tab set, 100  
 host commands, 71  
 host response, 37

## —I—

index, 100, 105, 124  
 interface, 32  
 italic, 112

## —K—

key attributes, 89  
 keyboard, 21, 39, 40, 41, 89, 92, 99, 100, 101, 102, 105,  
 108, 112, 113, 114, 115, 126  
 keyboard attributes, 90

## —L—

label, 1, 22, 23, 25, 26, 27, 31, 33, 37, 40, 41, 48, 85, 86,  
 89, 90, 91, 95, 96, 106, 108, 125  
 layering, 28  
 line style, 49  
 line style attributes, 98, 110  
 link trend graph, 101  
 list report, 105  
 load font list into character set, 101  
 local response, 37, 85  
 lock keyboard, 101  
 logo, 22, 23, 25, 26, 86

## —M—

menu, 1, 21, 28, 37, 38, 39, 40, 90, 91, 94, 95, 115, 124  
 menu attribute, 90  
 menu item attributes, 90  
 menu list report, 98  
 menu responses, 91  
 module state default, 101  
 module state restore, 101  
 module state save, 101  
 multidrop, 101  
 multidrop receive control, 101  
 multidrop transmit control, 101, 102  
 multiple touch mode, 43  
 multi-state, 23, 31, 32, 33, 35, 83, 86, 107  
 multi-state buttons, 31

## —N—

new line, 81, 101  
 next button page, 98  
 next line, 34, 81, 102  
 normal video, 102  
 numeric keyboard, 39

## —P—

pattern fill, 102  
 periodic function, 102  
 periodic functions, 72  
 pop graphic cursor position, 103  
 print buffer request, 102

print buffer response, 102  
print control, 97, 102  
print cursor line, 102  
print formfeed, 102  
print input buffer, 103  
print screen, 81, 102  
printer, 5, 97, 98, 102, 103  
printer input, 103  
printer port configure, 103  
printer status, 5, 98  
push graphic cursor position, 103

## —Q—

query button free space, 35  
query set/reset state, 103  
QWERTY keyboard, 39, 41

## —R—

raise button to top, 104  
read color attributes, 97  
read data/display list, 93  
read graphic pixels, 104  
read system error counters, 104  
real-time clock, 71  
rectangle, 2, 47, 49, 50, 51, 87, 109  
rectangle draw, 104  
repeat previous touch report, 104  
report, 1, 33, 43, 44, 45, 69, 70, 71, 81, 82, 93, 95, 96, 97,  
98, 99, 102, 104, 105, 106, 113, 114, 123, 124, 125,  
126  
report enter touches, 104  
report exit touches, 104  
report graphic objects, 104  
report multiple touches, 105  
report tracking touches, 105  
request button-list report, 105  
reset, 6, 52, 53, 54, 95, 98, 103, 105, 106  
reset graphics character attributes, 105  
reset graphics string attributes, 105  
reset graphics write mode, 105  
reset mode, 98, 105  
reset to initial state, 105  
reset touch modes, 105  
response-style keyboard, 21, 39  
reverse index, 105  
reverse video, 105, 109, 112  
reverse video (screen), 105  
RS485, 101, 102  
RTC display format, 111  
RTC time, 112

## —S—

screen reset on receive, 106  
screen saver, 105  
screen saver off, 105  
screen saver time-out, 106

scroll up, 106  
scrolling, 3, 24, 26, 28, 29, 33, 85, 104, 106, 116, 125  
self-check, 69, 70  
self-check function, 69  
selftest error counter, 99  
selftest mode, enter, 95  
set active button, 106  
set active window, 106  
set alpha write mode, 106  
set bad beam beep time, 106  
set bad beam report, 106  
set button attribute, 29  
set button page, 30, 106  
set button repeat time, 86  
set button state, 107, 125  
set button system options, 107  
set button timeout, 107  
set character set, 106  
set character size, 106  
set color attributes, 27, 108  
set fill pattern attribute, 109  
set graph cursor type, 109  
set graphic cursor position, 109  
set graphic object marks, 110  
set graphic write mode, 110  
set graphics character attributes, 109  
set graphics string attribute, 110  
set graphics write mode, 51  
set line style attributes, 110  
set RTC display format, 111  
set RTC time, 112  
set text attributes, 112  
set text blink rate, 112  
set text cursor style, 112  
set touch screen sensitivity, 112  
set user-defined fill pattern, 50, 112  
set user-defined line style, 49, 113  
set video mode, 26, 28, 113  
set video page, 113  
set/reset functions, 72  
set-up key, 113  
sleep, 113  
store font list, 113  
system error reporting, 113  
system operation commands, 69  
system self-test, 114  
system status request, 114  
system status response, 114

## —T—

tabs, 5, 81, 100  
text attributes, 112  
text blink, 112  
text button area, 91  
text cursor position, 99  
text cursor style, 112  
text mode, 33  
text terminal operation, 1

text underline, 112, 114  
toggle set/reset state, 114  
touch, 1, 22, 23, 25, 31, 32, 33, 35, 41, 43, 44, 45, 69, 70,  
81, 84, 85, 90, 91, 99, 100, 104, 105, 107, 112, 114  
touch dimensions, 99  
touch inquiry, 44  
touch modes, 43  
touch position, 99  
touch reports, 44, 69, 104, 105  
touch screen sensitivity, 69, 112  
track mode, 43  
transmit 7-bit control code sequences, 114  
transmit 8-bit control codes, 114  
transmit 8-bit request state, 114  
trend graph, 82, 91, 101

## —U—

undraw all menus, 115  
undraw button, 115  
undraw keyboard, 115  
undraw menu, 115  
unlink trend graph, 115  
unlock keyboard, 115  
update graphic object, 115

user interface tools, 21  
user-definable character sets, 55  
user-define fill pattern, 112  
user-defined line style, 113  
utilities and demos disk, 6, 21, 22, 24, 25, 38, 39, 40, 48,  
50

## —V—

vector draw, 115, 116  
video mode, 33, 99, 113, 125  
video page, 99, 113  
VT100, 3  
VT220, 3

## —W—

windows, 22, 23, 24, 25, 26, 28, 29, 30, 33  
write graphic pixels, 116  
write text string, 29, 116

## —X—

XON/XOFF enabled, 116



# Warranty

DEECO SYSTEMS, A DIVISION OF COMPUTER DYNAMICS INCORPORATED, A SUBSIDIARY OF TOTAL CONTROL PRODUCTS, INC.

Deeco Systems  
7640 Pelham Road, Greenville, SC 29615  
Phone: (864) 627-8800

## WARRANTY

Deeco Systems' products are warranted for a period of two years from the date of purchase against all defects in materials and workmanship provided they are properly used and not modified by non-Deeco personnel. Subassemblies and items not manufactured by Deeco (power supplies, disk drives, etc.) are warranted for the period established by their original manufacturer. Deeco will repair or replace the product, provided that it is returned promptly to Deeco at the owner's expense. Prior to returning a component or subsystem, the purchaser must obtain a Return Material Authorization number (RMA#) from Deeco. All board level products are shipped in an antistatic bag to prevent damage to the electronic components due to electrostatic discharge. Failure to use the bag in shipment will VOID the warranty. No other warranty is expressed or implied.

## DISCLAIMER

Deeco makes no representation or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Further, Deeco reserves the right to revise the prices or specifications and to make any changes from time to time in the contents hereof without obligation of Deeco to notify any person of such revisions or changes.

---

## To Our Customers:

It is our intention to provide you with accurate and useful information about our product. Although the information is correct to the best of our knowledge, we cannot assume responsibility for inaccuracies within the manual.

We request that you inform us of any errors found, areas difficult to understand or suggestions to improve this manual. Please fill out the bottom portion (using additional sheets if necessary) with your comments and return it to Deeco.

Thank you.

---

Name:	_____	Deeco Systems
Company:	_____	7640 Pelham Road
Address:	_____	Greenville, SC 29615
	_____	Phone: (864) 627-8800
Phone:	_____	
Product Type:	_____	Card Serial No. _____

## COMMENTS: