QTERM-K65 USER'S MANUAL
REVISION 5

QSI CORPORATION
2212 South West Temple #50
Salt Lake City, Utah 84115-2648
USA

Phone 801-466-8770
Fax 801-466-8792
Email info@qsicorp.com
Web www.qsicorp.com

Manual 0054-04
02395E1 - Printed in USA

QSI LIMITED WARRANTY

QSI Corporation warrants that its products are free from defects in materials and manufacturing for a period of one year from date of shipment. QSI further warrants that its products, when shipped, comply with the explicit product specifications contained in QSI's currently effective sales literature within the tolerances therein stated, determined in accordance with QSI's standard test procedures.

QSI will, at its option, repair or replace any product which does not comply with this warranty. The purchaser shall bear all costs of shipping products for which warranty coverage is sought to QSI's factory in Salt Lake City, Utah, USA. Purchaser shall pay all costs of removal and reinstallation of the product.

This warranty is voided by: (1) Any modification or attempted modification to the product done by anyone other than an authorized QSI employee (other than user adjustments performed in accordance with QSI product manuals); (2) If power supplied to the product exceeds the stated tolerance, or if the product is operated in environments exceeding the product's stated tolerance for temperature, humidity, moisture, radiation or electromagnetic fields; or (3) Any abuse, negligent handling or misapplication of the product.

THIS CONSTITUTES THE SOLE WARRANTY MADE BY QSI. THERE ARE NO OTHER WARRANTIES, EXPRESSED OR IMPLIED, WHICH EXTEND BEYOND THOSE DESCRIBED HEREIN OR TO ANYONE OTHER THAN THE ORIGINAL PURCHASER, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL QSI BE LIABLE FOR ANY INCIDENTAL OR CONSEQUENTIAL DAMAGES, OR FOR THE INFRINGEMENT OF ANY PATENT RIGHTS OR THIRD PARTY RIGHTS DUE TO THE USE OF ITS PRODUCTS.

QSI REPAIR POLICY

Warranty Products — Any product under warranty may be returned to QSI, shipping prepaid, and will be repaired and returned to the customer. QSI will pay the freight to return the product to the customer.

Non-Warranty Products — Any product QSI manufactures which is not under warranty may be returned to QSI, shipping prepaid, for an estimate of repair cost. QSI will provide the customer with such an estimate, and, at the customer's option, will either repair the product or return it to the customer. QSI may, at its option, either invoice the customer or require prepayment of repair and return shipping charges. The customer is liable for return shipping costs whether or not s/he requests QSI to repair the product.

Non-warranty Repair Charges — Charges include parts, labor and freight costs. Labor is charged at $100 per hour; the minimum labor charge is $100. Products returned for warranty repair that have no malfunctions will incur the $100 minimum labor charge.

RETURN/RESTOCK POLICY

Stock QSI Products (holsters, cables, QTERM-J001, etc.) that have *not* been used may be returned within 90 days of our invoice date for a 15% restocking charge. Semi-Custom Products that have *not* been used may be returned within 90 days of our invoice date for a restocking charge of 50%. *(Please note that all QSI terminals ordered with a worksheet are semi-custom, since the units are assembled to the worksheet specifications on receipt of your order.)* Custom Products are non-returnable.

In some instances QSI may offer salvage value for unused custom products; call for more information. QSI reserves the right to *not* accept any returns of any product after 90 days of our invoice date, or to accept them with a higher restocking charge or for salvage value only.

FCC Notice

This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the FCC rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generate s, uses and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications . Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at his own expense.

FOREWORD

The QSI Corporation QTERM-K65 is a rugged operator interface for industrial applications. It features a touch screen and a lighted, graphical, monochrome display. The terminal uses a specialized, object-oriented protocol (called Visual Terminal Protocol™) to generate and manipulate images on the display.

Chapter 1 **Introduction to Object-Oriented Terminal Protocols**. This section provides a brief orientation to the concepts of Object-Oriented Programming (OOP) and its application for graphical operator interfaces.

Chapter 2 **Basic Operation.** Instructions on connecting a QTERM-K65 terminal to a host computer and performing simple communications are described in this chapter.

Chapter 3 **Visual Terminal Protocol™.** This chapter provides a detailed description of Visual Terminal Protocol and discusses how to use it to control the QTERM-K65 terminal.

Chapter 4 **QTERM-K65 Hardware.** This chapter discusses the QTERM-K65 hardware, including dimensional drawings, interface specifications, connector pin assignments and environmental specifications.

Appendix A **ASCII Chart.** This is a true 7-bit ASCII chart, along with mnemonic definitions.

Appendix B **Font and Bitmap Data Structures**. This section provides detailed information on the binary format for Font and Bitmap data used in the QTERM-K65.

Appendix C **Visual Terminal Protocol Command Summary.** This is an abbreviated summary of VTP™ software commands.

Appendix D **Example VTP Commands**. Some example command sequences to create and modify objects are given in this section.

Appendix E. **Binary Data Communication**. This section describes the format for transmitting binary data to the QTERM-K65. It also describes the procedure for upgrading the terminal firmware through the serial interface.

Appendix F **Keypad VTP Commands**. This section describes the VTP commands for creating and manipulating key objects.

Appendix G **Auxiliary Serial Port**. This section describes VTP extensions that support use of the auxiliary serial port.

CONTENTS

CHAPTER 1. INTRODUCTION TO OBJECT-ORIENTED TERMINAL PROTOCOLS 1
1.1 What Are Objects? 1
1.1.1 A Sample Text Object 1
1.1.2 A Sample Gauge Object 2
1.1.3 Compass Needle 2
1.1.4 What Displayable Objects Can We Use? 2
1.2 Non-Displayable Objects 3
1.3 Summary 3

CHAPTER 2. BASIC OPERATION 5
2.1 Factory Configuration 5
2.2 Apply Power 5
2.3 Power-On Setup 5
2.4 Connect Communications Lines 6
2.4.1 EIA-232 6
2.4.2 EIA-422 7
2.4.3 5-volt Buffered 7
2.5 Communicate 7

CHAPTER 3. VISUAL TERMINAL PROTOCOL™ 9
3.1 Overview 9
3.2 Definitions 9
3.3 Issuing Commands to the QTERM-K65 10
3.4 Object Types and Attributes 10
3.4.1 Text Object 10
3.4.2 Font Object 11
3.4.3 Bitmap Object 11
3.4.4 TouchKey Object 12
3.4.5 Gauge Object 13
3.4.6 Line Object 13
3.4.7 Form Object 13
3.4.8 VT52 Object 13
3.4.9 Polyline Object 14
3.4.10 System Object 15
3.4.11 Key Object 15
3.5 Issuing Commands to the QTERM-K65 15
3.6 Commands for Managing and Using Objects 16
3.6.1 Create Object or Form 16
3.6.2 Delete Object 16
3.6.3 Select Object 16
3.6.4 Set Object Parameter 16
3.6.4.1 Alignment 16
3.6.4.2 Animation 17
3.6.4.3 Appearance Attributes 17
3.6.4.4 Binary Data 17
3.6.4.5 Border 18
3.6.4.6 Data 18
3.6.4.7 First Point 18
3.6.4.8 Font 19
3.6.4.9 Frame 19
3.6.4.10 Global Data 19

3.6.4.11 Justify 19
3.6.4.12 Key Press String 19
3.6.4.13 Key Release String 20
3.6.4.14 Key Release Attribute 20
3.6.4.15 Level 21
3.6.4.16 Modify 21
3.6.4.17 Name 21
3.6.4.18 Number of Data Points 21
3.6.4.19 Offset 21
3.6.4.20 Orientation 22
3.6.4.21 Point Definition 22
3.6.4.22 Position 23
3.6.4.23 Text Data Query 23
3.6.4.24 Second Point 23
3.6.4.25 Size 23
3.6.5 Enable Object 24
3.6.6 Disable Object 24
3.6.7 Query Object 24
3.6.8 Delete Form 24
3.6.9 Select Form 24
3.6.10 Enable Form 24
3.6.11 Disable Form 24
3.6.12 Query Form 25
3.6.13 Key Lockout 25
3.7 VTP Commands for the VT52 Object 25
3.7.1 Auto Linefeed Mode 25
3.7.2 Auto Scroll Mode 25
3.7.3 Auto Wrap Mode 25
3.7.4 Emulation Mode Enable 25
3.7.5 Tab Stop Settings 26
3.7.6 Operation of the VT52 Object 26
3.7.7 Supported VT52 Escape Sequences 26
3.7.7.1 Cursor Up - ESC A 26
3.7.7.2 Cursor Down - ESC B 26
3.7.7.3 Cursor Right - ESC C 26
3.7.7.4 Cursor Left - ESC D 26
3.7.7.5 Clear Screen - ESC E 26
3.7.7.6 Set Text Mode - ESC F Mode 26
3.7.7.7 Cursor Home - ESC H 27
3.7.7.8 Alternative Set Cursor Position - ESC I # * 27
3.7.7.9 Erase to End of Screen - ESC J 27
3.7.7.10 Erase to End of Line - ESC K 27
3.7.7.11 Query Character - ESC W 27
3.7.7.12 Query Cursor Position - ESC X 27
3.7.7.13 Set Cursor Position - ESC Y # * 27
3.7.7.14 Exit VT52 Emulation Mode - ESC < 27
3.7.7.15 Set Cursor Mode - ESC b # 27
3.8 Interaction with the System Object 27
3.8.1 Adjust/Set Backlight 27
3.8.2 Set/Adjust Contrast 28
3.8.3 Beep 28
3.8.4 Autoexec String 28
3.8.5 Key Beep Control 28
3.8.6 Touch Screen Enable 28
3.8.7 Key Repeat 29

3.8.8 Query Error Buffer 29
3.8.9 Query System Object 29
3.8.10 Firmware Version Query 29
3.8.11 Available RAM Query 29
3.8.12 Display Capabilities Query 29
3.8.13 Set Speaker Note 29
3.8.14 Set Beep Duration 29
3.8.15 Speaker Tone On/Off 30
3.8.16 Set Internal Timer Rate 30
3.8.17 System Reset 30
3.8.18 Flow Control Enable/Disable 30
3.8.19 Write Terminal Settings to ROM 31
3.8.20 Perform Touch Screen Calibration 31
3.8.21 Power-On Setup Enable/Disable 31
3.9 Creating and Storing Object Configurations 31
3.9.1 Write Objects to ROM 31
3.9.2 Read Objects from ROM 31
3.9.3 Clear All Objects from RAM 32
3.9.4 Restore Terminal Defaults 32
3.10 Default Object Configuration 32

CHAPTER 4. QTERM-K65 HARDWARE 33
4.1 QTERM-K65 Enclosure 33
4.2 Communication Interfaces 33
4.2.1 EIA-232 Interface 33
4.2.2 EIA-422 Interface 34
4.2.3 5-volt Buffered Interface 34
4.3 Other QTERM-K65 Hardware 34
4.3.1 Display 34
4.3.2 Touch Screen 34
4.3.3 Standard Touch Screen Legend 34
4.3.4 Memory 34
4.3.5 Speaker 35
4.4 QTERM-K65 Specifications 35
4.4.1 Environmental Specifications 35
4.4.2 Operational Specifications 35

APPENDIX A. ASCII CHART 37

APPENDIX B. FONT AND BITMAP DATA STRUCTURES 39

APPENDIX C. VISUAL TERMINAL PROTOCOL COMMAND SUMMARY 41

APPENDIX D. EXAMPLE VTP COMMANDS 47

APPENDIX E. BINARY DATA COMMUNICATION 51
E.1. Binary Data Definitions 51
E.2. Firmware Upgrade 51
E.3. CRC Source Code 52
E.4. References 52

APPENDIX F. KEYPAD VTP COMMANDS 55
F.1. Key Object Overview 55
F.2. Key Object VTP Commands 55
F.2.1. Create Key Object 55

F.2.2. Key Press String....55
F.2.3. Key Release String....55
F.2.4. Key Position....55

APPENDIX G. AUXILIARY SERIAL PORT....57
G.1. Auxiliary Serial Port VTP Commands....57
G.1.1. Set Serial Port Parameters....57
G.1.2. Send a String to the Auxiliary Serial Port....57

CHAPTER 1.

INTRODUCTION TO OBJECT-ORIENTED TERMINAL PROTOCOLS

Historically, operator interface terminals have used character-based displays to present simple text to the user. As an alternative, modern display technology now offers low cost graphical display devices. This provides greater capabilities and flexibility to the interface. However, standard terminal protocols (such as Digital Equipment Corporation's VT52™ and VT100™ protocols) have little or no support for controlling a graphics display. The programmer is required to control the entire display at the pixel level.

To alleviate this need, QSI Corporation has developed Visual Terminal Protocol™ (VTP™). VTP uses Object-Oriented Programming (OOP) techniques to simplify control of the graphical display. This chapter briefly discusses some basic concepts of OOP and how QSI has applied these concepts in VTP.

1.1 What Are Objects?

In VTP, an object can be Text, a Bitmap, a Line, a Polyline, a Gauge, a TouchKey, a Key or a Font. Some objects are displayable (such as Text or a Bitmap) while others (such as a Font) are used by the displayable objects. Each object has certain properties that are associated with it. Properties of displayable objects might include their position on the screen, their size and the data that they display. Some examples should illustrate these ideas more clearly.

1.1.1 A Sample Text Object

Suppose that we want to display the character string "K65" on the terminal display. Without a graphical terminal protocol, we resort to sending raw pixel data to the terminal. Using a 5-pixel by 7-pixel font, this information can be transmitted in about 21 bytes, presuming that we have the pixel patterns for these characters stored on the host.

If we wish to move the object to a different location on the screen, we must erase the old pixel patterns (21 bytes) and resend the pixel pattern to the new location (21 bytes). Of course, the host must keep track of where each byte is sent to get the proper pattern.

Using VTP, the characters can be displayed using a Text object. There are three steps to using an object in VTP:

1) create the object
2) define the object's properties
3) display the object on the screen

The properties for the Text object include the pixel location of the field, how to align the field around that pixel, the field size and the font to use in displaying text. The text data must also be provided. Default values will be used if the properties are not set explicitly.

First we issue a command to create the Text field (about six bytes), then define the text data as "K65" (seven bytes). We will use the defaults for text size, position and font. Finally, we "enable" the object, which tells the terminal to paint the object to the screen (two bytes).

With three simple commands and eight fewer bytes, we have created and displayed text on the screen. By using a built-in font, we need not concern ourselves with pixel patterns at all.

Better yet, to move the text we merely issue a command to change the position property of the Text object (about ten bytes; see Figure 1-1). If we decide to use a larger font, we just change the font property of the Text object and specify the name of desired font. The savings in host processing time and complexity are significant.

Perhaps this example is unconvincing. After all, many graphical displays have built-in functionality for emulating character displays. Some even have multiple built in fonts. An example using a non-text object is therefore appropriate.

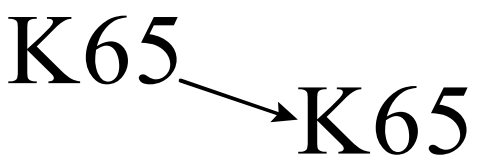


Figure 1-1. Moving text on the screen. A simple VTP command replaces manual erasure and redraw of the text.

1.1.2 A Sample Gauge Object

Suppose that instead of text we want to display a bar graph indicating the level of some quantity known to the host. Without VTP, we manually define an area of the screen that will represent the bar graph. The host must determine which pixels must be lit to represent the gauge border and fill. If the level changes, the fill must be recalculated and the pixel data must be retransmitted.

VTP offers a Gauge object for the described purpose. The properties of a Gauge object include the level of fill and the direction of increasing fill (or orientation) in addition to the normal attributes of size, position and alignment. We send commands to create the object, define its properties and display it. When the represented quantity changes, one simple command from the host will update the level of the gauge (see Figure 1-2).

1.1.3 Compass Needle

Now suppose we want to display a simple compass on the screen. We need the character "N" to indicate north and a line to represent the compass needle. The "N" is straightforward; we handle it as text.

Drawing the line is more challenging. Without VTP, we need to determine the endpoints of the line, interpolate every pixel between these endpoints and transfer this pixel data to the terminal. Every time the needle moves, we must transfer data to erase the line and repeat the process. This is time consuming work for the host.

With VTP, we create a line object and define its endpoints. As the needle moves, the host sends a single command to move the endpoint at the tip of the needle. Although the host must calculate the location of this endpoint, the terminal does the rest of the work (see Figure 1-3). No complicated display driving routines need be written or debugged.

Hopefully, it is now evident that VTP objects hide the non-essential details of displaying information from the host. The procedures that actually draw objects on the screen are indirectly called by accessing the properties of each object. In the OOP world, this is called *encapsulation.* VTP uses encapsulation to hide the complex details of drawing pixels on the screen.

Another OOP principle found in VTP is *polymorphism*. This means that the same function can be applied to different objects. Each object is intelligent enough to know how that function applies to it. For example, the procedures needed to resize a Text object are quite different from the procedures used to resize a Gauge object. Nonetheless, the same VTP command is used to resize either object. The object itself determines the procedures used to make the change.

1.1.4 What Displayable Objects Can We Use?

These examples illustrate text, gauges and lines as displayable VTP objects. In addition, a Bitmap object is provided to display arbitrary pixel patterns. Bitmap objects may include multiple "frames"; a single VTP command will cause a bitmap to animate by cycling through these frames at a user-specified rate. The four displayable objects provide tremendous flexibility in designing the user interface for the terminal.

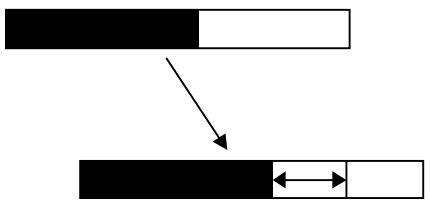


Figure 1-2. Gauge Objects. Moving the gauge or changing the level are simple VTP operations; they are difficult procedures when directly driving the display.

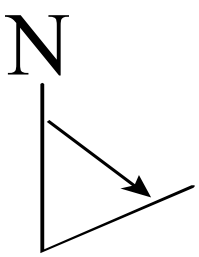


Figure 1-3. A Compass Needle. Redefining the endpoint property of the line with VTP moves the needle. No interpolation of pixels is necessary.

1.2 Non-Displayable Objects

VTP includes four types of objects that are not directly displayed on the screen.

A TouchKey object is used to define the actions that the terminal takes when a specific area of the screen or touch-active legend is pressed. This can include sending data to the host or executing a series of commands locally.

If the terminal includes a keypad, the Key object is used to define the actions taken by the terminal when a specific key is pressed. This object is similar to the TouchKey object in most respects.

A Font object defines the pixel patterns used by Text objects to display characters. The K65 terminal provides three built-in fonts for immediate use. With the Font object, the user is free to define additional fonts, including non-character fonts such as an icon library.

A Form object is used to group other objects together into screens of information. This allows the grouped objects to be displayed or erased together with a single command.

1.3 Summary

VTP addresses the inadequacies of standard character-based terminal protocols. The OOP features of VTP allow the terminal programmer to generate and manipulate complex graphical objects through a uniform and simple-to-use protocol. Most of the detailed work required to display objects is performed by the terminal, freeing the host to attend to more important duties. In other words, VTP saves you work; it gives you the power of a flexible graphical user interface without the traditional cost.

CHAPTER 2.

BASIC OPERATION

Connecting the QTERM-K65 terminal to the COM port on a PC computer is a simple procedure. Only four steps are required to power the unit and begin communications:

- apply 9 to 32 VDC to the V+ line (see Table 2-1)
- use *Power-On Setup* to set the display contrast, baud rate and data format
- connect the transmit, receive and ground lines
- transmit to and receive from the terminal

2.1 Factory Configuration

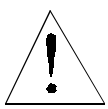
The QTERM-K65 terminal is shipped from the manufacturer with certain preconfigured options. These include the serial communications protocol (e.g., EIA-232, EIA-422, 5-volt Buffered) and the protocol data format (e.g., number of data bits, parity).

The default configuration is EIA-232 communications with 57,600 baud, eight data bits, no parity and one stop bit. If this is the desired communications configuration for your terminal, it will not be necessary to use the Power-On Setup feature before communicating with the device.

2.2 Apply Power

Table 2-1 shows the pin assignments for the power, ground and communication lines. Connect your DC power supply to the appropriate two pins. You must supply 9 to 32 volts DC at about 450 mA at 25 °C (the current will vary depending on the input voltage and ambient temperature). If power is properly applied to the unit, the firmware version number will be displayed for approximately 1 second, and an audible beep will be emitted from the terminal.

WARNING: Power supplied to the QTERM-K65 must be from an SELV power source and should have a current limit on its output of 5 Amperes. The supply must provide a minimum of 9 volts DC and be limited to a maximum of 32 volts DC. Limiting may be inherent to the supply or may be provided by supplementary overcurrent devices. If the QTERM-K65 does not respond, or exhibits abnormal behavior on power up, disconnect power and contact QSI for technical support.



2.3 Power-On Setup

The *Power-On Setup* procedure is used to configure the QTERM-K65's display contrast, baud rate and data format (see Table 2-2). For keypad-only devices, power-on setup is not available. Please contact QSI for setting these parameters.

To perform the *Power-On Setup* follow these steps:

- Remove power from the QTERM-K65 terminal.
- Press your finger onto and hold down the upper left corner of the touch screen area on the screen legend, and apply power to the terminal. Hold this area down until the power-on setup screen appears on the display (see Figures 2-1 and 2-2), then release your finger.
- The power-on setup screen shows labeled touch key areas on the terminal screen. These labels indicate the function of each key in configuring the terminal.

Table 2-1. QTERM-K65 Pin Assignments.

Pin Number	EIA-232/5-volt Buffered Function
1	No Connect
2	Tx (QTERM-K65 to host)
3	Rx (host to OTERM-K65)
4	No Connect
5	Ground
6	No Connect
7	CTS
8	RTS
9	V+ (power to QTERM-K65)

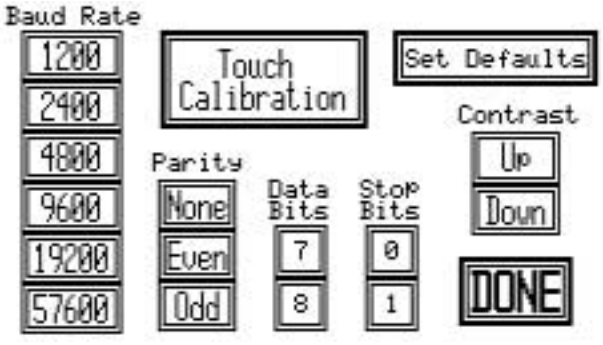


Figure 2-1. Power-On Setup Screen for 240x128 Display.

- The display colors of certain touch areas are inverted to indicate the current values for baud rate and data format. The current display contrast is self-evident.
- The touch screen response may be calibrated by pressing the *Touch Calibration* button. A new screen appears with instructions to touch the crosshairs as they appear on the screen. A normal key press with your finger (like pressing a key on your computer keyboard) is sufficient. After each set of crosshairs is touched, a new set appears in a different location of the screen. Four sets of crosshairs will appear in this manner. After all four sets have been touched, the calibration sequence is complete. The terminal will now return to the power-on setup screen.
- Set the baud rate by pressing the desired baud rate value on the display. The current baud rate should become highlighted on the display as each key is pressed.

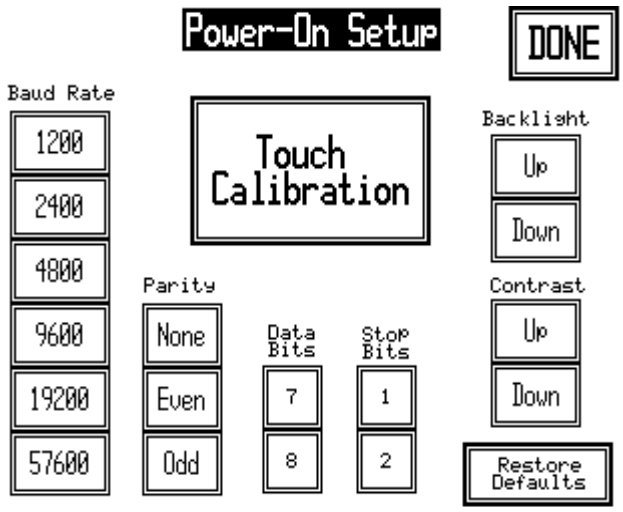


Figure 2-2. Power-On Setup for 320x240 Display.

Table 2-2. Available Baud Rates & Data Formats.

Available Baud Rates					
57,600		19,200		9,600	
4,800		2,400		1,200	
Available Data Formats*					
8n2	7n2†	8e2	7e2†	8o2	7o2†
8n1	7n1†	8e1	7e1†	8o1	7o1†

*Format is: data-parity-stop bits
†Format does not support binary data transmission

- Set the desired number of data bits, number of stop bits and parity in a similar manner. The current data format becomes highlighted as each key is pressed.
- The display contrast is adjusted by pressing the contrast *Up* and *Down* keys until a suitable contrast is achieved.
- The display backlight is adjusted by pressing the backlight *Up* and *Down* keys until a suitable level is achieved. (This option is only available on the 320x240 display).
- The *Set Defaults* button (*Restore Defaults* on the 320x240 display) may be used to reset the terminal to the default values for display contrast, baud rate, data format and touch calibration.
- After the desired features have been selected, press the *DONE* key in either the lower right area of the terminal (240x128 display) or the upper right area (320x240 display). This will cause the QTERM-K65 to exit power-on setup mode and save the settings to nonvolatile memory.

2.4 Connect Communications Lines

The QTERM-K65 can be factory-configured as either an EIA-232 device, an EIA-422 device or a 5-volt Buffered device.

2.4.1 EIA-232

A QTERM-K65 configured with the EIA-232 interface may be directly connected to the COM port on a personal computer (PC). This connection is made through a standard DB9 male-DB9 female communications cable. DC power must be supplied to the unit through pin 9 of the connector. The cable to make this connection must be supplied by the user. If the computer COM port uses a 25-pin EIA-232 connector, a DB9 male-DB25 female cable or a DB9-DB25 adapter must be included.

2.4.2 EIA-422

The EIA-422 interface is not directly compatible with a PC COM port. If required, here are some ways to make this connection:

- Use an interface device such as QSI's QCOM-2 unit which provides an EIA-232 port for the PC and an EIA-422 port for the QTERM-K65 terminal. The QCOM-2 also provides power to the terminal.
- There are numerous third-party 232-to-422 converters available. They range from external modules that plug into your COM port to ISA-bus daughterboards that plug into your motherboard and replace the EIA-232 COM ports with EIA-422 ports.

2.4.3 5-volt Buffered

The 5-volt Buffered option allows a host to communicate to the QTERM-K65 using standard TTL level (0 to 5 volt) signals. The pinout for the 5-volt Buffered option is summarized in Table 2-1.

2.5 Communicate

Set your host software communication parameters to the same values to which you set the QTERM-K65 terminal during power-on setup.

At this point, the QTERM-K65 unit is ready to accept VTP commands from the PC. Note that although commands may be received and understood by the terminal, nothing will be displayed on the screen until displayable objects are created and enabled. See Chapter 3, "Visual Terminal Protocol™," for details.

A simple test to verify communications can be performed by sending a beep command (sB;) to the terminal. The terminal should respond with a short audible beep, which is identical to the beep heard when power is applied to the unit.

QSI Corporation distributes software tools called KTerm32 and FormBuilder, which allow you to communicate with the QTERM-K65 from a PC. You can also use almost any commercial communications software such as ProComm, QMODEM or HyperTerminal to communicate with the QTERM-K65 terminal.

CHAPTER 3.

VISUAL TERMINAL PROTOCOL™

3.1 Overview

The QTERM-K65 uses an object-oriented communications protocol called Visual Terminal Protocol (VTP). VTP uses ASCII characters with a simple protocol to give you complete object-level control of a graphics-based operator interface. This chapter describes the VTP protocol and shows you how to achieve the full potential of your graphical QTERM-K65 terminal.

Object-oriented VTP gives you a tremendous amount of flexibility with the QTERM-K65, while simplifying the task of programming the "look" of the graphics display. If you are unfamiliar with object-oriented concepts, we suggest you read Chapter 1 before proceeding.

3.2 Definitions

This section includes definitions of VTP terms. Italicized words refer to other definitions in the list.

Alignment - This object attribute determines the alignment of the object relative to the position pixel (see *Position*). The alignment is set using keywords; the first keyword indicates horizontal position, while the second keyword indicates vertical position. Legal keywords are

Horizontal Keywords	Vertical Keywords
Left*	Top*
Middle	Middle
Right	Bottom

*Default

Attributes - Individual properties set by the *host* to define an object. For example, a text object has attributes that define its text data, font, size, position on the screen, etc.

Object Type - A predefined object description. VTP includes types such as a "text type" which defines the attributes of a text object (position, font, text, etc.) and "bitmap type" which defines the attributes of a bitmap object (width, height, position, bit pattern, etc.). A complete list of object types and their associated attributes can be found in Section 3.4.

Command - An instruction (two or more bytes) using VTP syntax is used to communicate with the QTERM-K65 terminal. Commands are sent from the *host* to the QTERM-K65 through a serial interface. All VTP commands are terminated with a semicolon (;).

Configuration - The collection of all forms and objects that are programmed into the QTERM-K65 to define the user interface to the terminal. The configuration can be considered the application software of the terminal. It is stored in the terminal's ROM and loaded into RAM after power-up.

Disabled - When a displayable object is disabled, it is erased from the display.

Enabled - When a displayable object is enabled, it is painted to the display, i.e. an enabled object is visible on the screen.

Form - A collection of *objects* that are grouped together, usually for simultaneous display.

Host - The computer that controls the QTERM-K65 by sending *commands* and interpreting *keystring* responses that occur through *user* interaction with the terminal.

Keystring - The character sequence that is sent to the *host* via the serial port when a specific area of the screen or touch-active legend is pressed.

Name - All *objects* or *forms* must be given a unique name when you define them. After they are defined, objects and forms are referred to by their names. Names can include any combination of letters and numbers. Spaces are also legal, but their use is not recommended. Names can be up to fifteen characters in length.

Object - A self-contained unit of data based upon one of the built-in types of objects provided by VTP. Each object has a unique set of *attributes* which determine its data and properties.

Position - An object attribute that tells the QTERM-K65 where to position the object on the screen. The host can move the object by changing its position attribute. The position attribute is always two numbers: the horizontal location (left to right) and the vertical location (top to bottom). For a 240x128 display, the upper left pixel of the display is 0,0; the lower right is 239,127. For a 320x240 display, the upper left pixel of the display is 0,0; the lower right is 319, 239. The default position for all objects is 0,0.

Size - This object attribute sets the size of the object box in pixels (except for VT52 objects where the size indicates rows and columns of characters). The *host* specifies the width and height of the object. Note that most objects displayed on the screen have an implicit one pixel wide border around the entire object.

User - The terminal operator who interacts with the *host* by viewing the terminal display and pressing predefined areas of the screen or touch-active legend.

3.3 Issuing Commands to the QTERM-K65

All VTP commands have the following format:

#param1,param2,...paramN;

where:

#	A single ASCII letter designating the specific command.
param1	Zero or more parameters for the specified command.
param2	Some parameters only use ASCII characters;
paramN	Others require binary data. The number, contents and format of parameters depend on which command is being issued.
,	Commas are required between multiple parameters.
;	A semicolon is required to terminate the command.

A detailed listing of commands and their associated parameters appears in Section 3.6.

3.4 Object Types and Attributes

There are several object types in VTP protocol. All user-defined objects must be based on one of these types. Below are descriptions of the object classes and their attributes. Specific VTP commands used to create objects and set their attributes are in Section 3.6.

3.4.1 Text Object

The text object is used to display text and character messages on the QTERM-K65 screen. A text object is a box in which characters are displayed. The box may have a border if desired. The characters are selected from a font object, which may be a host defined font or one of the built-in font objects. See the Font Object description for more details.

The text object has the following attributes:

Position - This sets the object position as described under Definitions.

Alignment - See the Definitions for a description of this attribute.

Size - This attribute is also described in the Definitions. The host may specify the width and height of the box in pixels. There is also an Autosize feature, which determines the box dimensions based on the amount of text and the selected font.

There are three possible configurations for size settings:

1) Explicitly set both width and height.

2) Explicitly set the width and Autosize the height.

3) Autosize both width and height (default).

If a line of text will not fit one line of the box, the text "wraps" to the next line at word boundaries. New line characters (ASCII 10 decimal) in the text will also force text to the next line. Setting the width to a fixed value of 1 causes the text to appear in a vertical column, one character wide.

The size attribute allows the host to create multi-line text fields and to customize the appearance of the text. By setting a border attribute (see below), the text object assumes a "windowed" appearance.

Data - The data attribute is used to specify the actual text that will appear in the text box. This text may include any characters that are defined in the selected font. The text data may be modified (rather than redefined) using

a Modify command. Characters may be added to or removed from the end of the string. Also, the current data in the Text object may be sent to the host by issuing a query command.

Font - The text font is selected with the font attribute. The host provides the name of the desired font when setting this attribute.

There are three built-in fonts provided with the QTERM-K65:

DF (default font), where characters are 5x7 pixels,
5x11Font, where characters are 5x11 pixels, and
8x16Font, where characters are 8x16 pixels.

The host may provide additional fonts by creating and defining new Font objects. If no font attribute is specified, the default (5x7 pixel) font will be used.

Border - This property may be used to create a customized border around the text box. The host sends a decimal value between 0 and 255, which represents a binary bitmask (8 bits wide). Each bit defines a "layer" of the border. Details and examples are given in Section 3.6.4.

Appearance Attributes - The appearance of the text box can be modified by setting these attributes, which include Inverse, Flash and Blink. Each of these attributes may be set to either *On* or *Off*.

Text is normally drawn to the QTERM-K65 display as white characters on a black background. Setting the Inverse property to *On* reverses this color scheme, causing the text field to appear as black letters on the white background of the text box. The Text object border is unaffected.

Setting the Flash attribute to *On* causes the text object to flash on and off, i.e. the text object will appear and disappear from the screen at a predefined rate. This rate is set in the System object (see Section 3.8). The default flash rate is 1 Hertz.

The Blink attribute is similar to Flash, except that Blink causes the text object to alternate between a normal display scheme (white letters on a black background) and an inverse color scheme. The rate of alternation is the same as the Flash rate described above.

The transparency of an object can also be set. Transparency determines whether the non-text pixels in the text box will display the pixels "behind" the object (transparent) or whether they will be painted black (opaque). This attribute also applies to the implicit one pixel border found on most objects. Objects are transparent by default.

If the Flash, Blink or Inverse attributes are set to *On*, the object automatically becomes opaque. In general, only one appearance attribute may be set to *On* at any given time. EXCEPTION: Inverse and Flash may be set to *On* simultaneously.

3.4.2 Font Object

Font objects contain binary data that define bitmapped patterns. Fonts are used to draw Text object characters or Bitmap object images on the display. The QTERM-K65 provides three built-in text fonts:

DF (default font), where character are 5x7 pixels,
5x11Font, where characters are 5x11 pixels, and
8x16Font, where characters are 8x16 pixels.

The text in bold indicates the name of the font. This is used to specify the Font attribute of a Text object. The host can define additional fonts by creating new font objects and providing the binary data to define the desired pixel patterns.

Font objects have just one attribute:

Binary Data - The binary pixel data are defined with this attribute. The QTERM-K65 uses a specific format for binary image data. This format consists of a 6-byte header followed by the actual pixel data (one bit per pixel). It is used for both Font and Bitmap objects; see Appendix B for details.

Font objects are very similar to Bitmap objects; they differ in that Font objects are global by default, and they cannot be displayed by themselves. Rather, they are displayed through Text and Bitmap objects.

3.4.3 Bitmap Object

The bitmap object is useful for displaying graphical images on the QTERM-K65 display. They may also contain multiframe animation sequences that cycle through frames at a predefined rate.

The following Bitmap object attributes are supported:

Position - See Definitions (Section 3.2).

Alignment - See Definitions (Section 3.2).

Binary Data - Refer to the corresponding Font object attribute.

Frame - Bitmap objects can support multiple "frames", or multiple images within the same object (All frames must be the same size). This feature is chiefly used for animating objects by rotating through frames at a prefined rate.

The frame attribute is used to set the currently displayed frame. The frames are numbered sequentially, beginning with Frame 0. The host may specify the actual frame number; alternatively, the host may increment or decrement to the next higher/lower frame. The default value of this attribute is Frame 0.

Global - In some applications, certain Bitmap objects may appear on multiple forms or in multiple locations on a form. To conserve memory and facilitate reuse of data, Bitmap objects have a Global attribute.

This attribute is very similar to the Font attribute for a text object. The global bitmap data is stored in a Font object on the ROOT form. The host then specifies the name of a Font which contains the desired image data. In this way, multiple Bitmap objects may share the image data.

Animation - This attribute is used to cause a bitmap to rotate through its frames at a predefined rate. The host provides a number indicating the duration that each frame is displayed, in 0.01 second increments. For example, specifying 50 would display each frame for 0.5 seconds. The frames are displayed in increasing order, beginning with frame 0. When the last frame has been displayed, the animation begins again with the first frame. To stop the animation, set this attribute to Off. The default value for new Bitmap objects is Off.

Appearance Attributes - All appearance attributes described with the text object are also supported for bitmap objects.

3.4.4 TouchKey Object

TouchKey objects are used to define the keystring that will be sent to the host through the serial port when an area of the screen or touch-active legend is pressed. Local keystrings which issue commands to the QTERM-K65 itself (as if the host had sent the commands) are also supported. Both local and host keystrings may be defined for the same TouchKey object.

TouchKey objects are associated with areas of the screen or legend by assigning a position and size attribute to the object (similar to placing text or a bitmap on the screen). The TouchKey object can associate host or local keystrings with either a press event or a release event.

The TouchKey object supports the following properties:

Position - See Definitions (Section 3.2).

Alignment - See Definitions (Section 3.2).

Size - See Definitions (Section 3.2). Also note that the minimum size for a TouchKey object is 8 pixels by 8 pixels. Attempts to set the width or height to less than these minimums will fail.

Key Press String/Key Release String - This attribute defines the host and/or local keystrings that are activated when the associated area of the screen or legend is pressed or released. Host keystrings are character strings that are sent to the host through the serial interface when the area is pressed or released. Local keystrings contain VTP commands, which are executed by the terminal when the area is pressed or released (as if they were sent by the host). The TouchKey object maintains completely separate keystrings for press and release events; either or both of these properties may be set.

The QTERM-K65 terminal uses a sampling algorithm to detect a "stable press" and a "stable release" on the touch screen. The algorithm detects a "stable press" after several samples of the touch location fall within approximately the same area. When a touch screen release is detected, the last several samples are examined to determine if they occurred in approximately the same area, indicating a stable release.

By dragging your finger across the touch screen immediately after touch or prior to release, it is possible to touch the screen without generating "stable touch" and/or "stable release" events. Key strings are only executed after stable events. If the Key Release attribute is OFF, it is possible to generate a touch event (executing a KeyPress string) without a corresponding release, and vice versa. Finally, if you touch the screen (generating a stable event), drag your finger to a new location, then release your finger (generating a stable event), the release will occur in the new location, possibly on a different TouchKey object or no TouchKey object at all.

If the Key Release attribute is set to ON, the terminal will always guarantee a key release of the last TouchKey object pressed no matter where the release is on the screen.

Border - This property may be used to create a customized border around the TouchKey area. The host sends a decimal value between 0 and 255, which represents a binary bitmask (8 bits wide). Each bit defines a "layer"

of the border. Details and examples are given in Section 3.6.4.

Auto-Inverse - This attribute determines whether the area defined by the TouchKey object will be inverted while the area is pressed, causing any visible objects in the TouchKey area to appear as black on a white background. This gives visual feedback to the user that the TouchKey object has been pressed. Enabling this feature on a TouchKey with a Key Press String that causes a form switch may lead to screen anomalies and is not recommended.

3.4.5 Gauge Object

Gauge objects are used to represent numerical data graphically as a bar. The data may range from 0 to 100, with 0 yielding an empty bar (a hollow rectangle) and 100 giving a full bar (a solid rectangle). For data between these extremes, the rectangle is filled in proportion to the number.

The gauge object supports the following properties:

Position - See Definitions (Section 3.2).

Alignment - See Definitions (Section 3.2).

Size - See Definitions (Section 3.2). The width of the Gauge object (or height for horizontal gauges) should be at least 5 pixels to properly display the gauge level.

Level - The numerical data determining the level of fill (0-100) on the Gauge object is set with this attribute.

Orientation - The Gauge object can be set to fill from the bottom, top, left or right side of the rectangle. The Orientation attribute sets this fill direction. Legal keywords for this attribute are *Up*, *Down*, *Left* and *Right*. These correspond to the direction of increasing fill as the value of the Level attribute increases.

Appearance Attributes - All appearance attributes described with the text object in Section 3.4.1 are also supported for gauge objects.

3.4.6 Line Object

A Line object is simply a straight line (one pixel wide) connecting any two points on the QTERM-K65 display. Multiple line objects can be used together to simulate more complex objects such as polygons, screen borders and line graphs.

The Line object has the following properties:

First Point - The x-y (horizontal-vertical) pixel coordinates for the first endpoint of the line.

Second Point - The x-y pixel coordinates for the second endpoint of the line.

3.4.7 Form Object

A Form object is a collection of other objects. The main purpose of the Form object is to organize other objects into manageable groups that can be displayed and erased easily. Usually a form is organized as a single screen of displayed objects. Multiple forms may be displayed simultaneously, if desired. The only form attribute is the current object (see Section 3.5).

3.4.8 VT52 Object

The VT52 object allows the K65 terminal to emulate a classic VT52 character display terminal. This object appears on the display as a window with a user-defined size, position and border. This window defines the virtual screen for VT52 terminal emulation. When the emulation mode for this object is enabled, characters sent from the host through the serial interface will be displayed inside the window. Standard VT52 escape sequences for moving the cursor, tabbing, erasing, etc. are supported. Section 3.7 provides detail and complete instructions for creating and using the VT52 object.

The VT52 object has the following attributes:

Position - This sets the object position as described under Definitions.

Alignment - See the Definitions for a description of this attribute.

Size - This attribute is also described in the Definitions. The host may specify the width and height of the box in *characters*, i.e. number of character rows and columns. Note that this differs from the size attribute for other objects.

Font - The font for the text appearing inside of the VT52 object window is selected with the font attribute. The host provides the name of the desired font when setting this attribute. See the description of the text and font objects for more information.

Border - This property may be used to create a customized border around the VT52 object window. The host sends a decimal value between 0 and 255, which represents a

binary bitmask (8 bits wide). Each bit defines a "layer" of the border. Details and examples are given in Section 3.6.4.

Emulation Mode Enable - This property enables VT52 terminal emulation for the object. All further communication from the host is directed to the object as VT52 terminal input until an *Exit VT52 Emulation Mode* escape sequence is sent to the terminal from the host. TouchKey objects will work normally while a VT52 object is in emulation mode, but only the host can exit the emulation mode by sending the correct escape sequence; emulation mode may not be disabled via keystrings.

Auto Wrap - This attribute determines what happens when the cursor moves past the end of the right edge of the VT52 object window. With auto wrap off, the cursor stays at the last position in the line. With auto wrap on, the cursor moves down to the first position in the next line.

If the cursor moves past the end of the last line in the window, and auto wrap is on, then the action depends on the setting of the auto scroll attribute. If auto scroll is off, the cursor will wrap to the first position of the current line, and the display window will not scroll. Otherwise, the display will scroll, and the cursor will return to the first position in the last line.

Auto Scroll - This property determines what happens when the cursor moves past the end of the last line in the VT52 object window. With auto scroll off, the cursor will stay in the last position. With auto scroll on, the display window scrolls (i.e. every line moves up, and the last line becomes blank), and the cursor moves to the first position in the last line.

Auto Line Feed - With auto line feed off, when a carriage return is received from the host, the cursor returns to the first position in the current line. With auto line feed on, the cursor moves to the first position in the next line, i.e. it acts as if both a carriage return and a linefeed had been received.

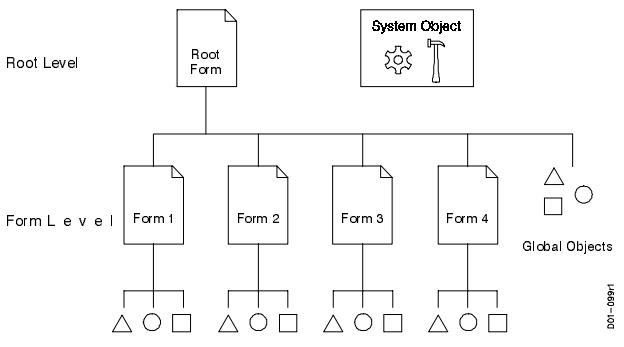


Figure 3-1. QTERM-K65 Object Hierarchy.

Tab Stop Setting - This property sets the character column locations of tab stops in the VT52 window. The leftmost column in the window is column 0. Tabs stops may be set in as many columns as desired.

3.4.9 Polyline Object

The Polyline object provides a convenient way to create multiple-line graphics. This object is constructed within a field of user-defined size and position. It is comprised of coordinates which specify the data points of the object's component line segments. Each data point is located relative to the lower left corner of the field (the center of a right-handed coordinate system).

The Polyline object has the following attributes:

Position - See Definitions (Section 3.2).

Alignment -See Definitions (Section 3.2).

Size - See Definitions (Section 3.2). A border attribute is not supported for the Polyline object; therefore, the one-pixel implicit border described in the Definitions Section does not apply. Polyline objects must extend at least three pixels in both dimensions.

Number of Points - This attribute defines the number of line data points in the Polyline object. At least two data points must be defined.

Offset - This attribute sets the location of the object's coordinate system origin within the field and consists of horizontal and vertical offsets. The default origin is in the lower left corner of the field. A positive horizontal offset moves the origin to the right, and vice versa. A positive vertical offset moves the origin up, and vice versa.

Point Definition - Data point locations can be specified three different ways:

1) A list of consecutive data point locations.

2) A linear fill of horizontal or vertical coordinates.

3) A list of consecutive horizontal or vertical coordinates.

A range of data points is specified for each of these methods. The range of data points must fall within the defined number of data points. Note that the second and third meth-

ods only define points for one axis, thus they must be used in conjunction with other methods to completely define a Polyline object's data points.

The first method uses a list of consecutive data points. All points in the given range must be specified. Each data point may be optionally preceded by an exclamation point. If a data point is preceded by this qualifier, the line segment joining the data point and the next one in the list will not be visible (i.e. not drawn on the display).

The second method uses a linear fill operation for the horizontal or vertical axis. Linear fill creates equally spaced data points over the given range on only one axis. Initial and increment values must be given for this operation. The initial value defines the horizontal or vertical position at the first data point. The data points are spaced according to the increment value.

The third method uses a list of consecutive horizontal or vertical coordinates. This method is identical to the first method except that points for only one axis are specified. Rules for the exclamation point qualifier apply to this method.

3.4.10 System Object

The System object is an interface between the host and the terminal hardware. Features such as the display contrast, display back light, speaker and key beep are accessed through the System object. The firmware version and available RAM may also be queried.

An error buffer is maintained in the System object. This structure is a FIFO (first in-first out) buffer that stores any error messages generated due to system errors. The buffer can hold up to 32 messages at a time. If the buffer becomes filled with 31 messages, the system logs an "Error buffer full" message in the last buffer slot.

The messages remain in the buffer until it is queried. Each query returns the next error message, which is then removed from the buffer. Additional errors may accumulate at the end of the buffer as space becomes available. If no error messages have been logged, the query returns a "No Errors" message. Also, the entire error buffer can be flushed with a single command from the host. In this case, the flushed error messages are not transmitted.

Details of the protocol for host interaction with the System object are described in section 3.8.

3.4.11 Key Object

See Appendix F.

3.5 Issuing Commands to the QTERM-K65

The hierarchy of objects used in the QTERM-K65 is illustrated in Figure 3-1. The hierarchy supports three levels of organization:

1) Root Level
2) Form Level
3) Object Level

The Root level exists as a form named ROOT, upon which all other forms are created. Global objects such as Fonts (including global bitmap images) live on the ROOT form. These objects are available to all other forms in the terminal. The System object also resides at Root level.

The second tier of the hierarchy is comprised of all Form objects. All forms are created on the ROOT form; the QTERM-K65 does not support "forms within forms."

At any given time, one form is always designated as *current*. The current form may be changed by executing a Create Form, Delete Form or Select Form command. When any of these commands are executed, the newly created or selected form becomes the current form. Rules for selection after a form is deleted are given in Section 3.6.8.

The current form is presently "in focus." That is, new objects are always created on the current form. Also, when a Select Object command is executed, only the current form is searched for the desired object. If objects on a different form are required, the host must first issue a Select Form command to change the current form.

NOTE: The enabled (visible) or disabled status of a form is independent of whether or not the form is current. Multiple forms may be enabled at any given time.

The third tier of organization contains all other objects (Text, Bitmap, TouchKey, Gauge, Line and Polyline). Font objects are created on the ROOT form; all other objects should be created on a second tier form.

Similar in concept to the current form, each form has a current object (which is presently "in focus"). All attribute change commands apply to this object. To change the current object, a Create Object, Delete Object or Select Object command must be executed. When a new object is created, it is automatically made current. Previously created objects are made current by executing a Select Object command. Rules for selection after an object is deleted are given in Section 3.6.2.

The ROOT form also has special significance with TouchKey objects. When an area of the screen is pressed, the

QTERM-K65 searches the current form for an enabled TouchKey object containing that area. If one is not found, the terminal then searches the ROOT form for an enabled TouchKey object containing that area. If no qualifying object is found on the ROOT form, the touchscreen press event is discarded. VTP applications can exploit this behavior to create "default" or global TouchKey objects that are active on all forms. Note that TouchKey objects on the current form will always override TouchKey objects (in the same area) on the ROOT form.

3.6 Commands for Managing and Using Objects

This section offers detailed explanations and syntax for the VTP commands to create, delete, select and manipulate forms and objects.

3.6.1 Create Object or Form

Syntax: A*type*,*name*;

where:
type = Type of object to be created
name = name for new object

Description: This command is used to create a new form or object. New objects are created on the current form. New forms and fonts are always created on the ROOT form. Object type designations are given in Table 3-1.

The object name can be any valid name (see Definitions) that is not already in use on the current form. A form name can be any valid name not currently assigned to another form. Attempting to create an object or form that already exists (i.e. the name is already in use) logs an error in the error buffer and aborts the create command.

Example: AT,my_text;

This creates a Text object named "my_text."

3.6.2 Delete Object

Syntax: B;

Description: The current object is removed from the current form. The new current object will be the next object in the form with respect to the order in which the objects were created (i.e. the next object that was created after the deleted object was created). If the deleted object was the first object to be created on that form, the next (or second) object becomes the current object and also the first object.

NOTE: Deletion of Font objects is not allowed. Font objects may only be removed from memory with the IF; or IA; command.

3.6.3 Select Object

Syntax: a,*name*;

where:
name = Name of object to be selected

Description: This command selects the object labeled "name" on the current form (making that object the current object). If no such object exists on the current form, an error is logged in the error buffer and the command is ignored.

Example: a,my_object;

This makes the object named "my_object" the current object on the current form, if an object named "my object" exists on the current form.

3.6.4 Set Object Parameter

Each parameter is explained separately. Note that not all parameters are applicable to all objects. The attributes that apply to each object type are listed in Section 3.4. If the attribute command is not applicable to the current object, the command is ignored.

3.6.4.1 Alignment

Syntax: bA,①,②;

where:
① = L, M or R
② = T, M or B
L = Left, M = Middle, R = Right,
T = Top, B = Bottom

Description: This command sets the object alignment as described in Definitions. The ① denotes a horizontal alignment keyword and ② denotes a vertical alignment keyword. The keywords may assume longhand or shorthand values as shown.

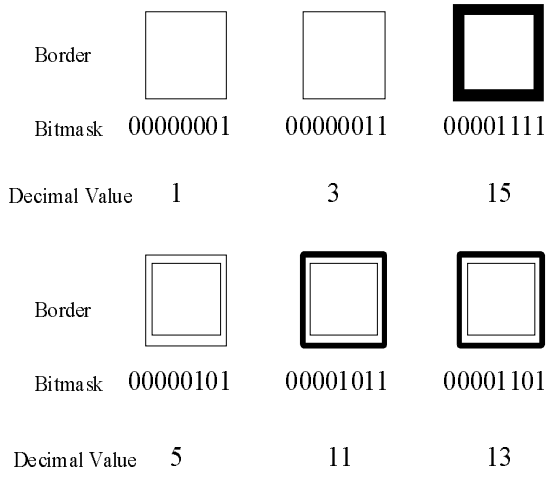


Figure 3-2. Typical Borders with Corresponding Masks.

Example: bA,M,B;

This sets the alignment of the current object to Middle (horizontal) and Bottom (vertical).

3.6.4.2 Animation

Syntax: bM,#;

where:
# = 1-65535 or O
O = Off

Description: For multi-frame Bitmap objects, this command causes the current object to "animate" by rotating through the frames (in increasing order), showing each frame for the specified period of time. In this case, # is the amount of time that each frame is displayed in units of 0.01 seconds. When the last frame has been displayed for the specified amount of time, frame 0 is redisplayed and the process begins again. Animation of the current object is terminated by passing this command with O for #.

Examples: bM,200;

This causes the current object to animate by displaying each frame of the bitmap for 2 seconds, then incrementing to the next frame.

bM,O;

This causes the current object to cease animating.

Table 3-1. Object Type Designations for VTP Create Commands.

Object Type	Shorthand Notation
Form	F
Font	O
Text	T
Bitmap	B
Touchkey	J
Gauge	G
Line	L
VT52	V
Polyline	P

3.6.4.3 Appearance Attributes

Syntax: bT,①,②;

where:
① = F, B, I, T
② = N or O
F = Flash, B = Blink, I = Inverse,
T = Transparency
N = On, O = Off

Description: This command sets the appearance attributes of Flash, Blink, Inverse and Transparency as explained in the Text object description. For TouchKey objects, the Inverse attribute sets the Auto-Inverse property. When this attribute is set, the key area colors will inverse when the key is pressed.

Example: bT,I,N;

This causes the Inverse attribute for the current object to be turned on.

3.6.4.4 Binary Data

Syntax: bB,<#bytes><~#bytes><bindata><CRC>;

where:
<#bytes> = A 4-byte header indicating the number of bytes in the <bindata> section. The header is binary and big endian (MSB first).
<~#bytes> = A 4-byte header section containing the complement of <#bytes>, such that the sum of quantities

<#bytes> and <~#bytes> equals zero.

<bindata> = The escaped binary data for the object.

<CRC> = A 2 byte cyclic redundancy check (CRC). See Appendix Appendix E. for more information.

Description: This command is used to define the data contained in a binary object (such as a Font or a Bitmap). The binary parts of this command are transmitted to the terminal one byte at a time. See Appendix Appendix E. for details.

Note that this command is not supported with communications formats that use only 7 data bits.

3.6.4.5 Border

Syntax: bR,*mask*;

where:
mask = 0-255

Description: The border attribute is used to create a customized border around a Text, TouchKey or VT52 object. The border is composed of up to eight "layers" or concentric boxes which lie on the perimeter of the text or key area.

A decimal representation of an eight bit mask is used to define the border. The least significant (rightmost) bit of the mask represents the innermost layer of the border, with additional layers represented by additional bits in the mask (from right to left). A "1" in a bit location indicates that the corresponding layer is painted (visible). Leading "0" bit values in the mask are ignored. By selecting an appropriate mask, thick borders and double borders can be created in many combinations (see Figure 3-2).

Examples: bR,1;

This places a single one-pixel-wide border around the current object (if the current object supports borders). The decimal mask of 1 represents a binary mask of 0000001, yielding the simple one-layer border.

bR,27;

In this case, the decimal 27 represents a binary mask of 00011011, giving a double border with a two-pixel-wide inner border and a two-pixel-wide outer border. The borders are separated by a one-pixel-wide space.

3.6.4.6 Data

Syntax: bD,*asciidata*;

where:
asciidata = the escaped ASCII data that will be contained in the object (i.e., text data, keystring, etc.)

Description: This command is used to provide text-based (non-binary) data definitions for the current object. For example, the text that will be displayed in a Text object is defined with this command.

Each comma and semicolon in ASCII data must be preceeded with a backslash character (\). This allows the terminal to ignore commas and semicolons in ASCII data when parsing the command.

Examples: bD,this is a bit of text;

This command places the characters "this is a bit of text" into the current object.

bD,hello\, how are you?;

This command uses data containing a comma.

3.6.4.7 First Point

Syntax: bF,#h,#v;

where:
#h = 0-319
#v = 0-239

Description: For Line objects, this command selects the x-y coordinates of the first endpoint of the line. #h is the horizontal pixel coordinate of the endpoint, and #v is the vertical pixel coordinate of the endpoint.

Examples: bF,100,50;

This sets the first endpoint of the current object to pixel 100,50.

bFirst,100,50;

Longhand equivalent of the first example.

3.6.4.8 Font

Syntax: bF,*fontname*;

where:
fontname=Name of font to be used.

Description: The Font attribute command selects the font for a Text or VT52 object. The font is selected from one of the built-in fonts or a host-defined font.

Example: bF,8x16Font;

This selects the built-in font named "8x16-Font" for the current object.

3.6.4.9 Frame

Syntax: bF,#;

where:
# = 0-65535, U or D
U = Up, D = Down

Description: This command selects the displayed frame for multi-frame Bitmap objects. Frames are numbered sequentially, beginning with 0. If a number is passed with this command, the corresponding frame is selected for display. If the number is greater than the number of frames in a bitmap, the terminal selects frame 0 by default.

If U is passed, the selected frame number is incremented (or returns to frame 0 if the last frame is currently selected). If D is passed, the frame number is decremented (or the last frame is selected if frame 0 is the current frame).

Examples: bF,4;

This selects frame 4 of the current object for display (if the current object is a Bitmap).

bF,U;
This increments the currently selected frame of the current object by one.

3.6.4.10 Global Data

Syntax: bG,*bitmapname*;

where:
bitmapname =Name of the global bitmap to be used.

Description: This command selects the global bitmap data to be used for this bitmap object. The global bitmap data is contained in a bitmap object created on the ROOT form. This global bitmap can then be displayed on any number of forms by creating a bitmap on the form and setting this attribute.

Example: bG,logo;

This selects the image in the bitmap object logo located on the ROOT form for display in the current object (if the current object is a Bitmap).

3.6.4.11 Justify

Syntax: bJ,①,②;

where:
① = L, M or R
② = T, M or B
L = Left, M = Middle, R = Right,
T = Top, B = Bottom

Description: For Text objects, this command sets the justification of the text within the text box. The ① denotes a horizontal alignment keyword and ② denotes a vertical alignment keyword.

Example: bJ,M,B;

This sets the justification of the current object (if it is a Text object) to Middle (horizontal) and Bottom (vertical).

3.6.4.12 Key Press String

Syntax: bD,*keystringdata*;

where:
keystringdata = the escaped ASCII data that defines the host keystring (to be transmitted to the host when the Touch-Key is pressed) and/or the local keystring (containing commands to be executed by

the terminal when the TouchKey is pressed).

Description: For TouchKey objects, *keystringdata* may contain host keystring data and/or local keystring data. Host keystrings are character strings that are sent to the host through the serial port when the screen or legend area is pressed. Local keystrings contain VTP commands that are executed by the terminal when the screen or legend area is pressed. VTP commands in local keystrings are executed as if these commands were sent by the host.

Host keystrings are defined by sending the desired character string as the *keystringdata* parameter. Each comma and semicolon in a host keystring must be preceded with a backslash character (\). This allows the terminal to ignore commas and semicolons in *keystringdata* when parsing the command.

Local keystrings are defined by sending VTP commands in the *keystringdata* parameter. The local keystring must begin with the characters \L. All commas and semicolons in the VTP commands contained in a local keystring must be preceded by a backslash (\).

If both a host keystring and a local keystring are desired for the same TouchKey object, they must both be defined using a single bD command. The host keystring must appear before the local keystring in the *keystringdata* parameter. All characters before the \L in *keystringdata* are part of the host keystring, while all characters after the \L are part of the local keystring.

Examples: bD,key area #1 was pressed;

For a TouchKey object, this defines a host keystring which sends the phrase "key #1 was pressed" to the host when the key associated with the key object is pressed.

bD,\LsB\;f\,FORM1\;;

For a TouchKey object, this defines a local keystring that sends the VTP commands sB;f,FORM1; to the terminal, as if the host had sent these commands. In response, the terminal will issue a beep and select the FORM1 form (if it exists) when the associated area is pressed.

bD,this key area was pressed\LsB\;f\, FORM1\;;

For a TouchKey object, this defines both the host keystring and the local keystring from the previous two examples. When the associated area is pressed, the host keystring phrase will be sent to the host and the VTP commands in the local keystring will be executed.

3.6.4.13 Key Release String

Syntax: bU,*keystringdata*;

where:
keystringdata = the escaped ASCII data that defines the host keystring (to be transmitted to the host when the TouchKey is released) and/or the local keystring (containing commands to be executed by the terminal when the TouchKey is released).

Description: For TouchKey objects, this attribute sets host and/or local keystrings for a TouchKey release event. In all other respects, this attribute is identical to the Key Press String attribute.

3.6.4.14 Key Release Attribute

Syntax: bT,U,①;

where:
① = N or O
N = On, O = Off

Description: The QTERM-K65 terminal uses a sampling algorithm to detect a "stable press" and a "stable release" on the touch screen. The algorithm detects a "stable press" after several samples of the touch location fall within approximately the same area. When a touch screen release is detected, the last several samples are examined to determine if they occurred in approximately the same area, indicating a stable release.

If the Key Release attribute is OFF, it is possible to generate a touch event (executing a

Key Press string) without a corresponding release. Finally, if you touch the screen (generating a stable event), drag your finger to a new location, then release your finger (generating a stable event), the release will occur in the new location, possibly on a different TouchKey object or no TouchKey object at all.

If the Key Release attribute is set to ON, the terminal will always guarantee a key release of the last TouchKey object pressed no matter where the release is on the screen.

Example: bT,U,N;

This will set the Key Release attribute to ON.

3.6.4.15 Level

Syntax: bL,#;

where:
= 0-100

Description: For Gauge objects, this command sets the fill level of the rectangle, i.e. the percentage (#) of the rectangle that will be filled.

Example: bL,57;

This sets the level of the current object to 57% (if it is a Gauge object).

3.6.4.16 Modify

Syntax: bM,①,②;

where:
① = A or R
A = Append, R = Remove
② = a character string (for append) or an integer (for remove)

Description: For Text objects, this command either appends characters to the end of the current data (Append) or removes characters from the end of the current data (Remove). For an Append command, the character(s) to be added are included in the ② field. For Remove, the integer in the ② field indicates the number of characters to remove from the end of the current text data.

Examples: bM,A, new ending;

This adds the character string "new ending" to the end of the current text object.

bM,R,4;

This removes the last four characters from the data in the current text object.

3.6.4.17 Name

Syntax: bN,newname;

Description: This command changes the name of the current object to "newname."

Example: bN,my object;

This changes the name of the current object to "my object."

3.6.4.18 Number of Data Points

Syntax: bN,#;

where:
= 2-2000

Description: This command defines the number of data points in the Polyline object.

Example: bN,6;

This sets the number of data points in the Polyline object.

3.6.4.19 Offset

Syntax: bO,#x,#y;

where:
#x = (-2000)-2000
#y = (-2000)-2000

Description: For a Polyline object, this command offsets the origin of the coordinate system from the bottom left corner of the object field. A positive #x moves the origin to the right, and vice versa. A positive #y moves the origin up, and vice versa. Note that the offset does not apply to the Polyline object field, but only to its coordinate system (i.e. the field does not change position).

Example: bO,-10,20;

This shifts the Polyline object's coordinate system origin to the left by ten pixels and up to twenty pixels from the bottom left corner of the object.

3.6.4.20 Orientation

Syntax: bO,①;

where:
① = U, D, L or R
U = Up, D = Down,
L = Left, R = Right

Description: For Gauge objects, this command sets the direction of increasing fill as the Level attribute is increased.

Example: bO,R;

This sets the direction of increasing fill for the current object (if it is a Gauge object) to Right.

3.6.4.21 Point Definition

Method 1:

Syntax: bD,#s:#e,[!]x0:y0 [!]x1:y1 [!]x2:y2 ... [!] xn:yn;

where:
#s = 0-(Number of Points-1) See Section 3.6.4.18.
#e = 0-(Number of Points-1) See Section 3.6.4.18.
#e ≥ #s
xn = (-2000)-(2000)
yn = (-2000)-(2000)

Description: This command defines the data points inside a Polyline object. Both #s (starting data point) and #e (ending data point) are zero based index values that define the range of data points to be created or updated. Data points are specified in coordinate pairs (xn=horizontal, yn=vertical) separated by colons. Note that there is one space separating each coordinate pair. If the coordinate pair is preceded by an exclamation point, then the line segment starting at that point and ending on the next point is not drawn.

Example: bD,0:4,10:1 20:2 !30:3 40:4 50:5

This defines the horizontal and vertical coordinates for the first five data points (0, 1, 2, 3 and 4). The line segment joined by the third and fourth data points is invisible.

Method 2:

Syntax: b[X][Y],#s:#e,F#b:#i;

where:
#s = 0-(Number of Points-1) See Section 3.6.4.18.
#e = 0-(Number of Points-1) See Section 3.6.4.18.
#e ≥ #s
#b = (-2000)-2000
#i = Value such that any coordinate does not exceed 2000 or -2000

Description: This command performs a linear fill of either the horizontal ('X') or vertical ('Y') data points. Both #s (starting data point) and #e (ending data point) are zero based index values that define the range of data points to be created or updated. 'F' denotes a fill operation. The term #b (begin value) sets the initial value that will be assigned to the data point at #s. The #b term is incremented by i# (increment value) and stored in the next data point in the defined range. This continues until #e is reached.

Example: bX,0:50,F20:2;

This assigns values to the horizontal coordinates in the first 51 points. The first x value is set to 20, the second to 22, and so on. The last x value is set to 70.

Method 3:

Syntax: b[X][Y],#s:#e,[!]p0 [!]p1 ... [!]pn;

where:
#s = 0-(Number of Points-1) See Section 3.6.4.18.
#e = 0-(Number of Points-1) See Section 3.6.4.18.
#e ≥ #s
pn = (-2000)-2000

Description: This command defines either the horizontal or vertical coordinates. Both #s (starting data

point) and #e (ending data point) are zero based index values that define the range of data points to be created or updated. Data points are specified as a list of either horizontal or vertical points (p0 ... pn). There is one space separating each coordinate. Note that if a data point has been defined with a preceding exclamation point, then the line segment joining that and the next data point will not be drawn.

Example: bY,3:8,2 5 !10 3 -10 -20;

This assigns values to the vertical coordinates in the fourth through ninth data points. The line segment joined by the sixth and seventh data points is invisible.

3.6.4.22 Position

Syntax: bP,#h,#v;

where:
#h = 0-319
#v = 0-239

Description: This command sets the x-y coordinates of the object position pixel (see Definitions). #h is the horizontal pixel coordinate, and #v is the vertical pixel coordinate. The object is aligned with this pixel according to the setting of the Alignment attribute. Objects that are positioned such that they hang off the screen are clipped at the screen's edge.

Example: bP,75,80;

This sets the position pixel of the current object to pixel 75,80.

3.6.4.23 Text Data Query

Syntax: bQ;

Description: For Text objects, this command causes the current data (text string) to be sent to the host through the serial interface.

3.6.4.24 Second Point

Syntax: bS,#h,#v;

where:
#h = 0-319
#v = 0-239

Description: For Line objects, this command selects the x-y coordinates of the second endpoint of the line. #h is the horizontal pixel coordinate of the endpoint, and #v is the vertical pixel coordinate of the endpoint.

Example: bS,200,80;

This sets the first endpoint of the current object to pixel 200,80.

3.6.4.25 Size

Syntax: bS,#w,#h;

where:
#w = a nonnegative integer
#h = a nonnegative integer

Description: This command sets the width and height (in pixels) of the current object. #w is the width and #h denotes the height. Objects that are sized such that they will not fit on the screen are clipped at the screen's edge.

The Text object has some autosizing features that are accessible with this attribute. If the letter A is entered in the #h field, the object width is set to #w and the height is autosized to the smallest value that will fit all of the text data in the object.

If the command bS,A; is entered, both the width and height of a Text object will be autosized. If no newline characters are found in the text data, all of the text data will be displayed on one line, with the width and height set to the minimum values that will display the data.

NOTE: For the VT52 object, the size parameters indicate the number of character rows and columns in the object, *not* the pixel width and height.

Examples: bS,15,30;

This sets the size of the current object to 15 pixels wide by 30 pixels high.

bS,50,A;

Set the width of a Text object to 50 pixels and autosize the height value.

3.6.5 Enable Object

Syntax: e;

Description: This command enables an object, causing it to be displayed on the screen (if it is a displayable object). The object must also be on a currently enabled form or it will not appear on the display. However, an object on a disabled form can be selected and enabled. It will then appear on the screen when that form is enabled.

TouchKey objects on the current form must be enabled to be activated by a press or a release event. Font objects are enabled at all times; if the current object is a Font object, this command has no effect.

3.6.6 Disable Object

Syntax: d;

Description: The current object is disabled, causing it to be erased from the screen. Enabled objects on disabled forms can also be selected and disabled. Disabled objects on a disabled form will not appear when the form is enabled.

Disabled TouchKey objects are not activated by a press or release event. Font objects are enabled at all times; if the current object is a Font object, this command has no effect.

3.6.7 Query Object

Syntax: q;

Description: The query command causes the terminal to send information about the current object to the host. This includes the object name, type and current setting of all object attributes.

Using this command is recommended for development only. Attributes may be added or enhanced in future versions of terminal firmware; the format of the terminal query response is not guaranteed.

3.6.8 Delete Form

Syntax: K;

Description: The current form is deleted. All objects that reside on the current form are also deleted. The new current form will be the next form on ROOT with respect to the order in which the forms were created (i.e., the next form that was created after the deleted form was created). If the deleted form was the last form to be created, the previous form becomes the current form.

3.6.9 Select Form

Syntax: f,*name*;

where:
name = Name of form to be selected

Description: This command selects the named form (making that form the current form). If no such form exists, an error is logged in the error buffer and the command is ignored.

Example: f,my form;

This makes the form named “my form” the current form, if such a form exists.

The QTERM-K65 supports the reserved form name !BACK to return to a previously selected form. The syntax for this command is

f,!BACK#;

where # is a single digit from 1 to 9. The digit indicates the previous form to select. For instance, a value of 5 selects the fifth most recently selected form. If the immediately previous form is desired, the # may be omitted (i.e., 1 is the default value).

3.6.10 Enable Form

Syntax: h;

Description: This command enables the current form, causing all enabled (displayable) objects on the form to be painted to the screen. Enabling a form does *not* automatically enable the objects on the form. More than one form may be enabled simultaneously.

3.6.11 Disable Form

Syntax: g;

Description: The current form is disabled, causing all objects on that form to be erased from the screen. Disabling a form does *not* disable all objects on that form.

3.6.12 Query Form

Syntax: Q;

Description: The query command causes the terminal to send information about the current form to the host. This includes the form name and object names and types.

Using this command is recommended for development only. Attributes may be added or enhanced in future versions of terminal firmware; the format of the terminal query response is not guaranteed.

3.6.13 Key Lockout

Syntax: L,①;

where
①= O or N
O = Off, N = On

Description: The key lockout command is similar to the Touch Screen Enable command; it enables/ disables the transmission of TouchKey host strings and the execution of TouchKey local strings. Keys are enabled when the lockout is Off (this is the default case). When the lockout is On, pressed keys are queued and processed when the lockout is removed (i.e. turned off). This is in contrast to the Touch Screen Enable command (see Section 3.2).

3.7 VTP Commands for the VT52 Object

This section details commands that are specific to the VT52 object.

3.7.1 Auto Linefeed Mode

Syntax: bL,①;

where:
① = O or N
O = Off, N = On

Description: The command enables or disables the auto linefeed mode of a VT52 object. See the VT52 object description for details on the effects of this mode.

Example: bL,O;

If the current object is a VT52 object, this command disables auto linefeed.

3.7.2 Auto Scroll Mode

Syntax: bC,①;

where:
① = O or N
O = Off, N = On

Description: The command enables or disables the auto scroll mode of a VT52 object. See the VT52 object description for details on the effects of this mode.

Example: bC,N;

If the current object is a VT52 object, this command enables auto scroll.

3.7.3 Auto Wrap Mode

Syntax: bW,①;

where:
① = O or N
O = Off, N = On

Description: The command enables or disables the auto wrap mode of a VT52 object. See the VT52 object description for details on the effects of this mode.

Example: bW,N;

If the current object is a VT52 object, this command enables auto wrap.

3.7.4 Emulation Mode Enable

Syntax: bE;

Description: This command enables VT52 terminal emulation for the object. All further communication from the host is directed to the object as VT52 terminal input until an *Exit VT52 Emu-*

lation Mode escape sequence is sent to the terminal from the host.

3.7.5 Tab Stop Settings

Syntax: bT,①:②:③:...;

where:
①,②,③... = #

Description: The command sets the locations of tab stops in the VT52 object window. The column numbers for the tab stops are provided as parameters (column number 0 is the first column in the VT52 window). More than one tab stop may be set by separating each column number with a colon. Sending this command deletes all tab stops set by previous Tab Stop commands.

Example: bT,3:7:10;

If the current object is a VT52 object, this command sets tab stops at column locations 3, 7 and 10.

3.7.6 Operation of the VT52 Object

The VT52 object is inactive until the *Emulation Mode Enable* command is sent to it. After this command is received by the object, all further communication from the host via the serial interface is processed by the object as if it were a VT52 terminal until the escape sequence to exit VT52 emulation mode is received by the QTERM-K65.

When VT52 emulation mode is entered for the first time, the cursor begins in the upper left hand corner of the window (i.e. row 0, column 0). If VT52 emulation mode is subsequently disabled and re-enabled, the cursor will remain at its location when VT52 emulation mode was disabled.

Printable characters received from the host while VT52 emulation mode is enabled will be printed at the cursor location in the VT52 object window. The cursor is automatically moved right one column as each character is printed. Behavior of the cursor at the right and bottom edges of the screen is determined by the state of the Auto Wrap and Auto Scroll attributes as previously described.

The following non-printable characters will produce the described effect when received by the VT52 object:

- **0x07** - Emit a beep from the speaker.
- **0x08** - Backspace. Move the cursor left by one column without deleting any characters.
- **0x09** - Tab. Move the cursor right to the next tab stop.
- **0x0A** - Linefeed. Move the cursor down one row.
- **0x0B** - Vertical Tab. Move the cursor down four rows.
- **0x0C** - Formfeed. Clears the VT52 object window.
- **0x0D** - Carriage Return. Moves the cursor to the beginning of the current line. If Auto Line feed is enabled, this character will move the cursor to the beginning of the next line.
- **0x7F** - Delete. Moves the cursor left and deletes the character at that screen location.

3.7.7 Supported VT52 Escape Sequences

In addition to the control characters described above, the behavior of the VT52 object may be controlled through the escape sequences, which are character sequences that begin with the escape character (**0x1B**). In this section, the escape character is denoted by the ⸘ symbol.

3.7.7.1 Cursor Up - ⸘ A

Moves the cursor up one line without changing its horizontal position. It has no effect if the cursor is on the first line.

3.7.7.2 Cursor Down - ⸘ B

Moves the cursor down one line without changing its horizontal position. It has no effect if the cursor is on the last line.

3.7.7.3 Cursor Right - ⸘ C

Moves the cursor right one space without changing its vertical position. It has no effect if the cursor is at the right-most position on the current line.

3.7.7.4 Cursor Left - ⸘ D

Moves the cursor left one space without changing its vertical position. It has no effect if the cursor is at the left-most position on the current line.

3.7.7.5 Clear Screen - ⸘ E

Clears the display and moves the display cursor to home (the left-most position in the top line of the display).

3.7.7.6 Set Text Mode - ⸘ F Mode

Sets the mode for text received after this escape sequence. The mode remains set until reset with another *Set Text Mode* sequence. The following modes are supported:

N - Normal text.
F - Flashing text (text appears and disappears at the system timer rate).

3.7.7.7 Cursor Home - ESC H

Moves the cursor to the home (top left) position on the VT52 object window.

3.7.7.8 Alternative Set Cursor Position - ESC I # *

Positions the cursor to the specified location. The command has the form *ESC I # **, where # sets the row and * sets the column. The row and column characters are determined by adding an offset of 64 (hex 40) to the desired row or column location. The corresponding ASCII character (see Appendix A) should be transmitted. The offset rows and columns to be designated by printable characters. For example, the string

ESC I B D

sets the cursor to row 2 (third row) and column 4 (fifth column).

3.7.7.9 Erase to End of Screen - ESC J

Erases from the current cursor position to the end of the screen. The cursor position is unchanged.

3.7.7.10 Erase to End of Line - ESC K

Erases all displayed characters from the current cursor position to the end of the line. The cursor position is unchanged.

3.7.7.11 Query Character - ESC W

Returns the ASCII character value for the character at the current cursor position.

3.7.7.12 Query Cursor Position - ESC X

Returns the cursor position as two ASCII characters. These two characters are defined in the same way as for the *Set Cursor Position* (ESC Y) command.

3.7.7.13 Set Cursor Position - ESC Y # *

Positions the cursor to the specified location. The command has the form *ESC Y # **, where # sets the row and * sets the column. The row and column characters are determined by adding an offset of 32 (hex 20) to the desired row or column location. The corresponding ASCII character (see Appendix A) should be transmitted. The offset rows and columns to be designated by printable characters. For example, the string

ESC Y ! $

sets the cursor to row 1 (second row) and column 4 (fifth column).

This command is very similar to the *Alternative Set Cursor Position* (ESC I) command. The ESC Y command has a lower offset, allowing more rows and columns to be addressed for a large VT52 window.

3.7.7.14 Exit VT52 Emulation Mode - ESC <

Exits VT52 emulation mode and returns the QTERM-K65 terminal to normal (VTP) command processing. The cursor remains in its current position.

3.7.7.15 Set Cursor Mode - ESC b

Determines the cursor attributes. The following cursor modes are supported:

U - Underline. Cursor appears as an underscore character.
N - No cursor. Cursor is not visible in the VT52 object window.

3.8 Interaction with the System Object

Commands for accessing the hardware features of the terminal through the System object are described in this section. Certain information, such as firmware version, available RAM and display capabilities may be obtained by querying the System object.

3.8.1 Adjust/Set Backlight

Syntax: sL,①;

where:
① = 30-255, O, N, U or D
O = Off, N = On, U = Up, D = Down

Description: The display backlight level is adjusted with this command. The backlight may be explicitly set to a certain level by passing a number to the System object. For a 240x128 display, the backlight range is from 30-255 (30 = backlight completely on, 255 = backlight completely off). Alternatively, the level may be adjusted to completely on by passing N or to completely off by passing O. Finally, the

backlight can be incrementally adjusted with U and D.

For a 320x240 display, the backlight range is from 50-255 (50 = backlight completely on, 255 = backlight completely off).

Examples: sL,150;

This sets the backlight to a level of 150.

sL,N;

This sets the backlight to completely on.

3.8.2 Set/Adjust Contrast

Syntax: sC,①;

where:
①= 0-255, D, L or R
D = Darker, L = Lighter, R = Reset

Description: The display contrast level is adjusted with this command. The contrast may be explicitly set to a certain level by passing a number to the System object (0 = contrast completely on, 255 = contrast completely off). Alternatively, the contrast is incrementally adjusted with D and L. The Contrast can be reset to factory default with R. The factory default for a 240x128 display is 135, the factory default for a 320x240 display is 145.

Examples: sC,150;

This sets the contrast to a level of 150.

sC,L;

This sets the contrast incrementally lighter.

3.8.3 Beep

Syntax: sB;

Description: This command causes the speaker to emit a single beep. The beep note and duration are set in the System object.

3.8.4 Autoexec String

Syntax: sA,$;

where:
$ = A string containing escaped VTP commands (see Description)

Description: The Autoexec string contains VTP commands which are executed after the terminal is reset or power is applied. This command allows the host to change the Autoexec string. Each comma and semicolon in $ must be preceeded with a backslash character (\). This allows the terminal to ignore commas and semicolons in $ when parsing the command. The Autoexec string is saved to non-volatile memory when the terminal receives a G; command.

Examples: sA,sC\,100\;sB\;;

This command sets the Autoexec string to: "sC,100;sB;." These commands will be executed after each terminal reset.

3.8.5 Key Beep Control

Syntax: sK,①,②;

where:
①= O or N (key down beep)
② = O or N (key up beep)
O = Off, N = On

Description: The short beep that is emitted whenever an enabled TouchKey area is pressed is called a key beep. This beep can be enabled or disabled with this command. An optional beep to indicate TouchKey release is also available.

Examples: sK,N,O;

This enables the key beep for TouchKey press events and disables the key beep for TouchKey release events.

3.8.6 Touch Screen Enable

Syntax: sP,①;

where
① = O or N
O = Off, N = On

Description: The touch screen is enabled and disabled with this command. If the touch screen is disabled, no key press or key release events are registered in the terminal. The key beep will

not sound if the touch screen is disabled. Refer also to the Key Lockout command.

Example: sP,N;

This enables the touch screen, allowing key events to take place.

3.8.7 Key Repeat

Syntax: sP,*delay*,*rate*;

where:
delay = 0-1000
rate = 0-1000

Description: This command sets the delay before the first key repeat and the rate of the repeats thereafter in 0.01 second increments. (e.g. 200 corresponds to two seconds).

A value of 0 in the delay field will turn off the key repeat.

Example: sP,100,50;

This causes a delay of 1 second after a TouchKey is pressed before repeating the key, then repeats the key every 0.5 seconds.

3.8.8 Query Error Buffer

Syntax: sE;

Description: This command returns the next error in the error buffer to the host. If the error buffer is empty, this command returns "No Errors." See Section 3.4.10 for information on the error buffer.

Optionally, the parameter F can be added to this command. This causes the entire error buffer to be cleared. No error message is returned with this option. The syntax is

sE,F;

3.8.9 Query System Object

Syntax: sQ;

Description: This command causes the terminal to send information on the current status of the System object. This includes the values in all System object attributes, available RAM, configuration checksum and serial communication parameters.

3.8.10 Firmware Version Query

Syntax: sV;

Description: This command causes the terminal to send the current firmware version to the host. The format for the response is "vX.XX", where an X represents a digit in the version number.

3.8.11 Available RAM Query

Syntax: sM;

Description: This command causes the terminal to send the current amount of used and free RAM. The response indicates the used and available RAM in bytes.

3.8.12 Display Capabilities Query

Syntax: sY;

Description: This command queries for information on the terminal display. The response indicates the type of display (graphical or character), and the dimension of the display in pixels.

3.8.13 Set Speaker Note

Syntax: sN,#;

where:
= 0-63

Description: This command sets the musical note (or frequency) generated by the speaker when a Beep or Speaker On command is issued. The note is set by passing an integer corresponding to the desired note, as shown in Table 3-2.

3.8.14 Set Beep Duration

Syntax: sD,#;

where:
= 1-65535

Description: The Beep Duration command sets the duration of the audible note when a Beep command is executed. # indicates the duration in

Table 3-2. Speaker Note Settings.

Number	Note	Number	Note
0	D2	32	G4#
1	D2	33	A4
2	D2	34	A4#
3	D2#	35	B4
4	E2	36	C5
5	F2	37	C5#
6	F2#	38	D5
7	G2	39	D5#
8	G2#	40	E5
9	A2	41	F5
10	A2#	42	F5#
11	B2	43	G5
12	C3	44	G5#
13	C3#	45	A5
14	E3	46	A5#
15	D3#	47	B5
16	E3	48	C6
17	F3	49	C6#
18	F3#	50	D6
19	G3	51	D6#
20	G3#	52	E6
21	A3	53	F6
22	A3#	54	F6#
23	B3	55	G6
24*	C4*	56	G6#
25	C4#	57	A6
26	D4	58	A6#
27	D4#	59	B6
28	E4	60	C7
29	F4	61	C7#
30	F4#	62	D7
31	G4	63	D7#

*Middle C

The number adjacent to each note indicates the octave (e.g. A4=note A, 4th octave).

0.01 second increments (e.g., 200 corresponds to a 2 second duration).

3.8.15 Speaker Tone On/Off

Syntax: sS,①;

where:
① = O or N

Description: The speaker is turned on and off with this command. Turning the speaker on causes it to emit the speaker note until it is turned off.

Example: sS,N;

This turns on the speaker note, causing a constant tone to be emitted until the speaker is turned off.

3.8.16 Set Internal Timer Rate

Syntax: sT,#;

where:
= 1-1000

Description: The internal timer controls timed functions such as the flash or blink rate of displayed objects. # indicates the time period of one clock tick in 0.01 second increments (e.g. 200 corresponds to a clock tick every two seconds). Each clock tick represents one period of the flash/blink cycle.

3.8.17 System Reset

Syntax: sR;

Description: This commands forces the terminal to undergo a warm boot cycle. Objects in RAM are lost and the configuration stored in ROM is restored.

3.8.18 Flow Control Enable/Disable

Syntax: sX,①;

where:
① = O, N or H

Description: This commands turns the XON/XOFF flow control operation on and off. N enables

XON/XOFF flow control and O disables it. For systems that are equipped with hardware flow control capability, the H enables hardware flow control. This uses the RTS and CTS lines to arbitrate transmission of serial data as described in Section 4.2.1.

Example: sX,N;

This command turns XON/XOFF flow control on.

3.8.19 Write Terminal Settings to ROM

Syntax: sW;

Description: This command causes the terminal specific settings such as display contrast and backlight to be written to ROM. These settings are automatically restored after system reset or power-on.

3.8.20 Perform Touch Screen Calibration

Syntax: sU,①;

where:
① = O or N

Description: This command causes the QTERM-K65 to enter touch screen calibration mode, as described in Section 2.3. If ① is set to on (N), the QTERM-K65 will send the string "~TchCal(0)~" to the host when the touch screen calibration is completed. When this setting is set to off (O), no string is transmitted at the end of calibration.

3.8.21 Power-On Setup Enable/Disable

Syntax: sJ,①;

where:
① = E, R or D

Description: This command controls access to the Power-On Setup facility during power-up. If this setting is set to E, the Power-On Setup facility is fully enabled. If set to R, the serial port settings (baud rate, parity, data bits and stop bits) are read only (i.e. they cannot be modified by Power-On Setup). If set to D, Power-On Setup is disabled and cannot be entered as described in Section 2.3.

Sending this command causes the terminal settings to be written to ROM as described in Section 3.8.19.

Example: sJ,R;

This sets Power-On Setup to be enabled with serial port settings as read only. It also causes the terminal settings to be written to ROM.

3.9 Creating and Storing Object Configurations

The configuration (see Definitions) may be saved to ROM after it has been programmed into the terminal. The configuration is automatically restored to RAM after system reset.

The terminal specific settings for baud rate, serial data format and display contrast level are also stored in ROM. These settings are stored separately from the configuration. The power-on setup feature automatically saves these settings to ROM. If these settings do not exist in ROM, the factory default settings are used instead (see Section 3.10).

3.9.1 Write Objects to ROM

Syntax: G;

Description: Executing this command causes all objects and forms currently in RAM to be written to ROM. Any configuration already in ROM is overwritten. This procedure takes approximately six seconds. During this time, the unit will not respond to commands.

3.9.2 Read Objects from ROM

Syntax: H;

Description: This command causes the configuration stored in ROM to be restored to RAM. This does NOT overwrite objects in RAM; all objects should be cleared from RAM before executing this command (see Section 3.9.3). The terminal executes this command automatically after reset or power-on. The current form and current objects are also retained, as are all object attributes. Therefore, the objects on the display when the Write Objects to ROM (G;) command is executed will be visible after the configuration is restored.

3.9.3 Clear All Objects from RAM

Syntax: IF;

Description: This command erases all objects and forms from memory but does not change the system configuration.

3.9.4 Restore Terminal Defaults

Syntax: IA;

Description: This command erases all objects and forms from memory and restores the system default objects. These objects are described in Section 3.10.

3.10 Default Object Configuration

The factory power-on default configuration of the QTERM-K65 is as follows:

baud rate	57,600
data format	8n1
XON/XOFF flow control	off
display contrast	135 (240x128 display) 145 (320x240 display)
display backlight	30 (240x128 display) 50 (320x240 display)
keypad	enabled
key beep	down, on; up, off
key repeat	0 (off)
key repeat delay	100 ms (10)
key beep time	40 ms (4)
key beep tone	5^{th} octave G (43)
beep duration	300 ms (30)
beep tone	6^{th} octave A# (58)
power-on setup	fully enabled
timer rate	1 sec (100)

The default objects are as follows:

Form: ROOT
    Font: DF
    Font: 5x11Font
    Font: 8x16Font

Form: MAIN
    No objects.

CHAPTER 4.

QTERM-K65 HARDWARE

4.1 QTERM-K65 Enclosure

The QTERM-K65 front panel is injection molded using black, glass-filled polyester. The touchscreen and legend form a water-tight seal around the interior of the panel. A gasket is provided to seal the outside edges of the terminal where it presses against the panel. The QTERM-K65 back panel is not waterproof, but moderate rain or splash should not cause harm to the unit.

The outside dimensions of the QTERM-K65 are shown in Figure 4-1. This figure also shows the location of the cable connector and the speaker aperture. The panel-mount QTERM-K65 is mounted directly onto your instrument or enclosure panel Figure 4-2 shows the mounting cutout required to install the terminal.

The panel-mount QTERM-K65 uses a DB9 connector exiting from the back of the housing. The pin assignments for the connector are shown in Table 2-1 (on page 5). Figure 4-3 shows the DB9 connector pin numbering.

4.2 Communication Interfaces

4.2.1 EIA-232 Interface

With proper cables and good grounding, the EIA-232 interface on the QTERM-K65 can communicate up to about 15 meters at 9,600 baud. At the terminal's maximum baud rate of 57,600 baud, cable length should be limited to about 10 meters.

The QTERM-K65 may be factory configured to support the EIA-232 modem-control lines (RTS or CTS). If this feature is available, hardware handshaking may be enabled by sending the **sX,H;** command. Handshaking between the host and QTERM-K65 may also be accomplished using software XON/XOFF commands. Binary transfers have been modified to allow the use of XON/XOFF; see Appendix Appendix E.. Flow control can be disabled (by sending the **sX,O;** command) if you do not wish to have it operate.

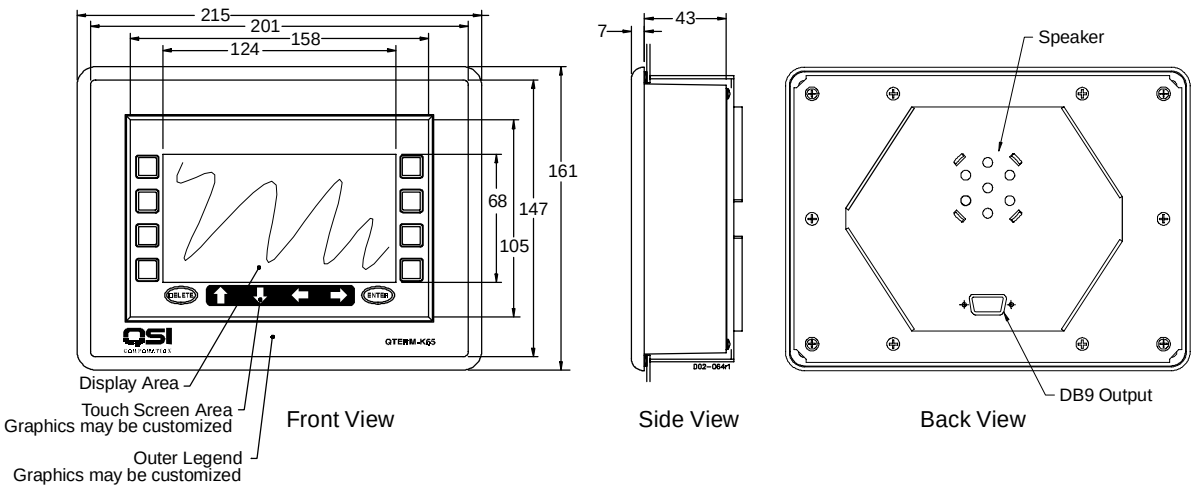


Figure 4-1. Dimensions of the Panel-Mount QTERM-K65 Terminal (mm).

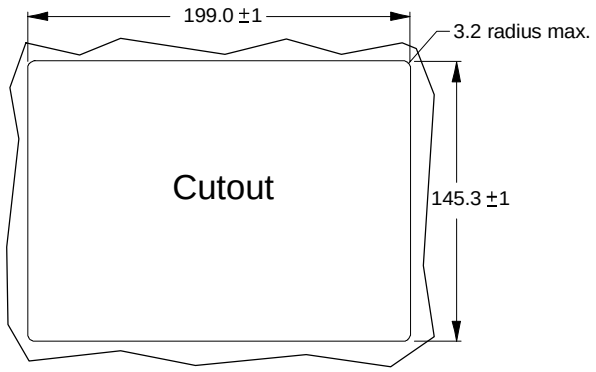


Figure 4-2. Cutout Dimensions for Panel-Mounting (mm).

4.2.2 EIA-422 Interface

With proper cables and grounding, the EIA-422 interface can communicate up to distances of about 1000 meters at 9,600 baud. The cable length should be shorter for higher baud rates.

The pin assignments for the panel mount QTERM-K65 with the EIA-422 interface are shown in Table 4-1.

4.2.3 5-volt Buffered Interface

When used to communicate with another 5-volt Buffered device, the QTERM-K65 can operate at distances up to about 5 meters. The advantage of the 5-volt Buffered QTERM-K65 is that it is lower power than the EIA-232 version and uses the same 5-volt logic levels as come directly out of a UART (i.e. the idle state is at 5 volts, and the active state is at 0 volts).

Although the signal levels are the same as at a UART, the transmit and receive lines are buffered and can be connected to live signals whether or not the QTERM-K65 is powered (turned on).

4.3 Other QTERM-K65 Hardware

4.3.1 Display

The QTERM-K65 can have a 240x128 or 320x240 graphical supertwist liquid crystal display (LCD) with a Cold Cathode Fluorescent Light (CCFL). Using bitmaps and text objects, any pixel pattern can be displayed.

Software commands for adjustment of the display contrast and back light settings are described in Section 3.8.

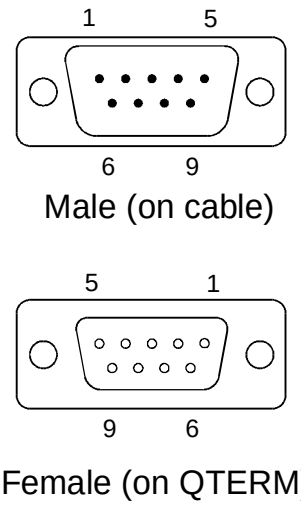


Figure 4-3. QTERM-K65 Connector Pin Numbering.

4.3.2 Touch Screen

The QTERM-K65 uses an analog resistive touch screen to provide flexible input capabilities to the terminal. Touch areas may be defined by the user in both the screen area and the surrounding interior legend (see Figures 4-4 and 4-5). These "keys" may be programmed to return keystrings to the host when pressed or released. Key areas may also trigger local commands that are executed by the terminal as if the host had sent them. Both of these features are programmed using TouchKey objects (see Chapter 3). A keypad may be included with the terminal as an option (see Appendix Appendix F.).

The key beep is enabled or disabled by the System Object, as described in Section 3.8.

4.3.3 Standard Touch Screen Legend

The standard QTERM-K65 touch screen legend is shown in Figure 4-4. This legend is customizable to your specifications. Please contact QSI for details. To assist in creating TouchKey objects for the buttons on the standard legend, the position and size of each button are listed in Tables 4-2 and 4-3.

4.3.4 Memory

The QTERM-K65 electronics include 512 kbytes of Flash ROM and 512 kbytes of RAM. The ROM contains the terminal bootloader, firmware and terminal-specific settings (256 kbytes), leaving 256 kbytes for object configuration.

New versions of terminal firmware may become available. The firmware is upgraded through the serial interface, elim-

Table 4-1. QTERM-K65 Pin Assignments.

Pin Number	EIA-232/5-voltBuffered Function	EIA-422 Function
1	nc*	Tx-
2	Tx	Tx+
3	Rx	Rx+
4	nc	nc
5	Ground	Ground
6	nc	Rx-
7	CTS	nc
8	RTS	nc
9	V+ (pwr)	V+ (pwr)
*nc = No Connect.		

inating any need to open the device and access the electronics. The procedure to upgrade the firmware is described in Appendix Appendix E..

4.3.5 Speaker

The QTERM-K65 includes a speaker used for key beeps and for software controllable beeping. The key beep and beep functions are controlled through the System object.

4.4 QTERM-K65 Specifications

4.4.1 Environmental Specifications

See Table 4-4 for the environmental specifications.

4.4.2 Operational Specifications

See Table 4-5 for operational specifications.

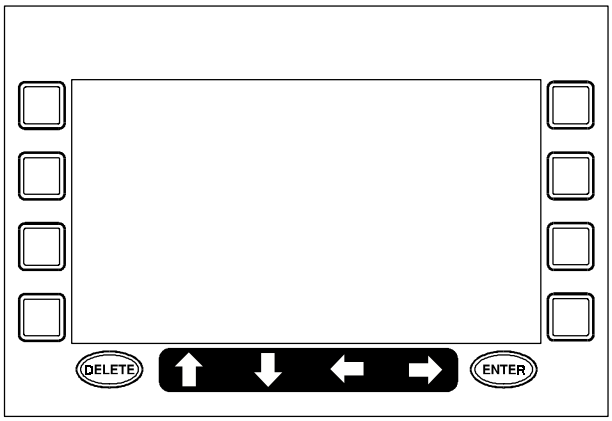


Figure 4-4. QTERM-K65 standard touch screen legend.

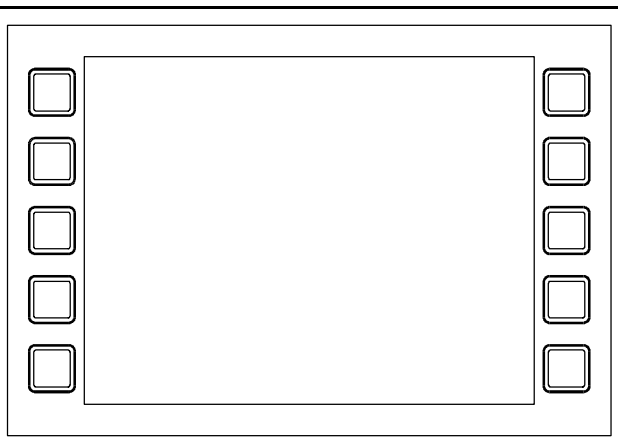


Figure 4-5. QTERM-K65 320x240 legend.

Table 4-2. Standard Legend Key Position and Size.

Key Locations	Position*	Size
Left side from top to bottom	-33,-3 -33,35 -33,71 -33,107	25,25
Right side from top to bottom	247,-3 247,35 247,71 247,107	25,25
Delete	-3,135	40,20
Enter	205,135	40,20
Up Arrow	45,135	20,20
Down Arrow	90,135	20,20
Left Arrow	130,135	20,20
Right Arrow	170,135	20,20
* Alignment Default:	Left, Top	

Table 4-3. 320x240 Legend Key Position and Size.

Key Locations	Position*	Size
Left side from top to bottom	-50,5 -50,55 -50,105 -50,155 -50,205	30,30
Right side from top to bottom	335,5 335,55 335,105 335,155 335,205	30,30
* Alignment Default:	Left, Top	

Table 4-4. Environmental Specifications.

Parameter	Value
Temperature	
Operating Range	-20 to 60 °C
Storage Range	-30 to 60 °C
Humidity	
Non-condensing	0-95%
Vibration	
Operating	4 g RMS 5 to 5000 Hz
Shock	20 g for 3 ms
Electrical Interference	
FCC part 15	Class A
Sealing	
fluids/dust	NEMA 4

Table 4-5. QTERM-K65 Operational Specifications.

Parameter	Value
Power Supply:	
Input Voltage Range	9 to 32 VDC
Input Current (typical), 9 VDC	450 mA @ 25°C
Input Current (typical), 32 VDC	135 mA @ 25°C
Input Current (240x128), 12 VDC	325 mA
Input Current (320x240), 12 VDC	295 mA
Touch Screen	
Type	Resistive
240x128 Display	
Viewing Area	126mm x 70mm
Dot Pitch	0.5mm x 0.5mm
320x240 Display	
Viewing Area	122mm x 92mm
Dot Pitch	0.36mmx0.36mm
EIA-232 Interface	
Maximum baud rate	57,600
Maximum distance (9,600 baud)	15 meters
Maximum distance(57,600 baud)	10 meters
EIA-422 Interface	
Maximum baud rate	57,600
Maximum distance (9600 baud)	1,000 meters
5-volt Buffered Interface	
Maximum distance	5 meters

APPENDIX A.

ASCII CHART

Least Significant Digit (hex)	Most Significant Digit (hex) 0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	`	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

0345R3

NUL = blank
SOH = start of header
STX = start of text
ETX = end of text
EOT = end of transmission
ENQ = enquiry
ACK = acknowledge
BEL = bell
BS = backspace
HT = horizontal tab
LF = line feed
VT = vertical tab
FF = form feed
CR = carriage return
SO = shift out
SI = shift in
SP = space
DLE = data link escape
DC1 = device control 1 (XON)
DC2 = device control 2
DC3 = device control 3 (XOFF)
DC4 = device control 4
NAK = negative acknowledge
SYN = synchronization
ETB = end of text block
CAN = cancel
EM = end of medium
SUB = substitute
ESC = escape
FS = file separator
GS = group separator
RS = record separator
US = unit separator
DEL = delete/rubout

APPENDIX B.

FONT AND BITMAP DATA STRUCTURES

Font objects and bitmap objects contain definitions of pixels that are on or off. This appendix explains the binary format for describing these pixel patterns. The file format has two sections: a six byte header followed by binary data for the pixel pattern (one bit per pixel).

The header is formatted as shown in Table B-1. The first byte is the number of characters (font) or frames (bitmap) in the object. The next four bytes define the character or frame width and height. These are both two-byte quantities ranging from 0 to 65535.

Table B-1. Font and Bitmap Data Header.

Byte Number	Font	Bitmap
1	Number of characters	Number of frames
2	Character width (high byte)	Frame width (high byte)
3	Character width (low byte)	Frame width (low byte)
4	Character height (high byte)	Frame height (high byte)
5	Character height (low byte)	Frame height (low byte)
6	ASCII Offset to first character	0x00

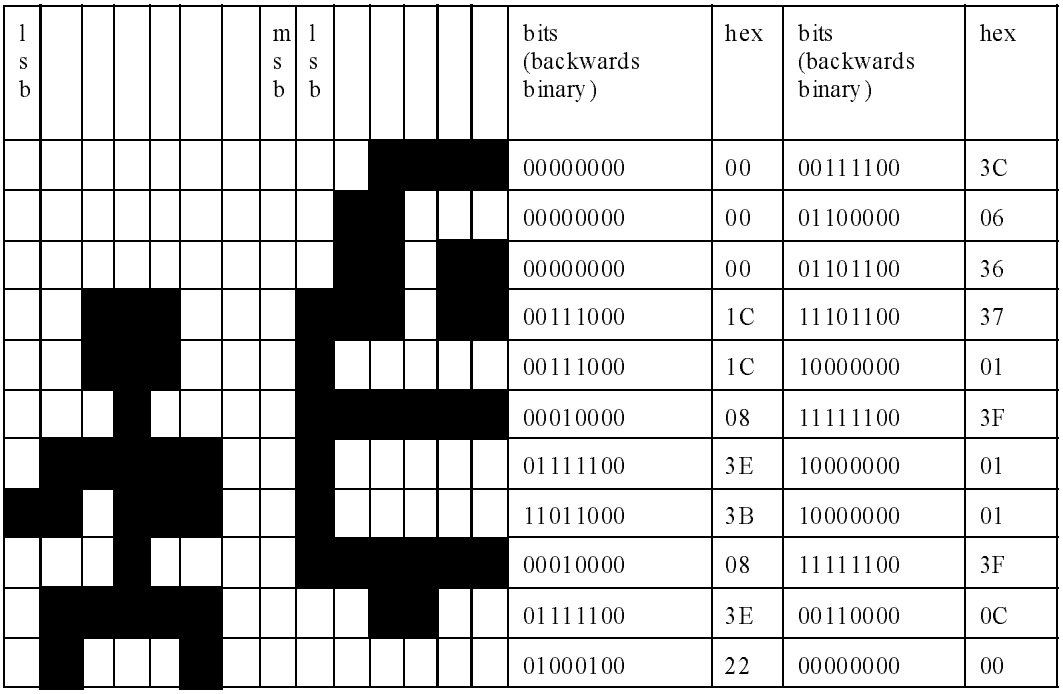
The sixth byte defines the ASCII decimal value used for the first character in the font. For example, if the first character was A, the offset would be 97 (see the ASCII chart in Appendix A). For this reason, font characters corresponding to ASCII characters should be defined in the order shown in the ASCII chart. This value is ignored for bitmaps (use 0x00, i.e. 00 hexadecimal).

Pixel data for defining fonts and bitmaps are sent as 8-bit bytes. Each byte represents eight pixels: a bit value of 0 turns the pixel off; a bit value of 1 turns the pixel on.

The bytes define the pixel pattern beginning with the top row of pixels, from left to right. The second row is then defined, from left to right, and so on. Bytes do not overlap to the next row, so if the width of the object is not a multiple of eight, the last byte in each row will contain data for less than eight pixels. The extra bits are ignored and should be filled with a value of 0.

Although the order of bytes defines pixels from left to right, the order of bits within a byte is reversed. That is, within each byte the least significant bit defines the leftmost pixel and the most significant bit defines the rightmost character. Figure B-1 shows an example pixel definition.

Since font and bitmap data are binary, the bB command should be used to send the data to the terminal. See Appendix Appendix E. for more information.



lsb							msb	lsb						bits (backwards binary)	hex	bits (backwards binary)	hex
														00000000	00	00111100	3C
														00000000	00	01100000	06
														00000000	00	01101100	36
														00111000	1C	11101100	37
														00111000	1C	10000000	01
														00010000	08	11111100	3F
														01111100	3E	10000000	01
														11011000	3B	10000000	01
														00010000	08	11111100	3F
														01111100	3E	00110000	0C
														01000100	22	00000000	00

Figure B-1. Pixel Definition Example.

APPENDIX C.

VISUAL TERMINAL PROTOCOL COMMAND SUMMARY

Table C-1. Object Management Commands.

Create Object	**A***type*,*name*; *type*=> O=Font T=Text B=Bitmap J=Touchkey G=Gauge F=Form L=Line K=Key V=VT52 P=Polyline
Delete Object	**B;**
Rename Object	**bN,***newname*;
Write Objects to ROM	**G;**
Read Objects from ROM	**H;**
Clean all forms and objects from RAM (ROM is intact)	**IF;**
Restore Terminal Defaults	**IA;**
Delete Form	**K;**
Query Current Form	**Q;**
Software Upgrade	**S,**<*hdr*><*bindata*><*CRC*>;
Key Lockout	**L,**①; ① = [**O**|**N**]

Table C-2. Object/Form Configuration Commands.

Select Object	**a,***name*;
Disable Object	**d;**
Enable Object	**e;**
Select Form	**f,***name*;
Select Previous Form	**f;!BACK#;** (# is optional)
Disable Form	**g;**
Enable Form	**h;**
Query Current Object	**q;**

Table C-3. Object Attribute Commands.

OBJECT	PARAMETERS	FORMAT	DEFAULTS (IF APPLICABLE)
Font	Binary pixel data	**bB,**<*hdr*><*bindata*><*CRC*>**;**	null data
Text	Alignment	**bA,**①**,**②**;** ① = [**L**|**M**|**R**] ② = [**T**|**M**|**B**]	L,T (Left,Top)
	Text (Alternate)	**bB,**<*hdr*><*textdata*><*CRC*>**;**	null data
	Text	**bD,***textdata***;**	null string
	Font	**bF,***fontname***;**	DF (5x7Font)
	Justification	**bJ,**①**,**②**;** ① = [**L**|**M**|**R**] ② = [**T**|**M**|**B**]	L,T (Left,Top)
	Modify (Append)	**bM,A,***textdata;*	
	Modify (Remove)	**bM,R,**#**;**	
	Position	**bP,**#**,**#**;**	0,0
	Text Data Query	**bQ;**	
	Border	**bR,***mask;*	0
	Size	**bS,**#**,**#**;**	A (Autosize)
	Autosize	**bS,**#**,**A**;bS,**A**;**	
	Attributes	**bT,**①**,**②**;** ① = [**F**|**I**|**B**|**T**] ② = [**O**|**N**]	T,N (Transparent,On)
	Rename Object	**bN**,*newname*;	
TouchKey	Alignment	**bA,**①**,**②**;** ① = [**L**|**M**|**R**] ② = [**T**|**M**|**B**]	L,T (Left,Top)
	Key Press String	**bD,***keystringdata***;**	null data
	Position	**bP,**#**,**#**;**	0,0
	Border	**bR,**#**;**	0
	Size	**bS,**#**,**#**;**	20,20
	Attribute (Auto-Inverse)	**bT,I,**①**;** ① = [**O**|**N**]	N (On)
	Attribute (Key Release)	**bT,U,**①**;** ① = [**O**|**N**]	O (Off)
	Key Release String	**bU,***keystringdata***;**	null data
	Rename Object	**bN**,*newname*;	
Key	Key Press String	**bD,***keystringdata***;**	null data
	Key Release String	**bU,***keystringdata***;**	null data
	Key Position	**bK,**#**,**#**;**	0,0
	Rename Object	**bN**,*newname*;	
Bitmap	Alignment	**bA,**①**,**②**;** ① = [**L**|**M**|**R**] ② = [**T**|**M**|**B**]	L,T (Left,Top)
	Pixel Data Definition	**bB,**<*hdr*><*bindata*><*CRC*>**;**	null data
	Frame to Display	**bF**,①**;** ① = [#|**I**|**D**]	0
	Global Bitmap	**bG,***name;*	null
	Animation	**bM,**①**;** ① = [#|**O**]	O (Off)
	Position	**bP,**#**,**#**;**	0,0
	Attributes	**bT,**①**,**②**;** ① = [**F**|**I**|**B**|**T**] ② = [**O**|**N**]	T,N (Transparent,On)
	Rename Object	**bN**,*newname*;	

Table C-3. Object Attribute Commands.

Gauge	Alignment	**bA,①,②;** ① = [**L**|**M**|**R**] ② = [**T**|**M**|**B**]	L,T (Left,Top)
	Level	**bL,#;** # = [**0-100**]	0
	Orientation	**bO,#;** # = [**U**|**D**|**L**|**R**]	U (Up)
	Position	**bP,#,#;**	0,0
	Size	**bS,#,#;**	8,3
	Attributes	**bT,①,②;** ① = [**F**|**I**|**B**|**T**] ② = [**O**|**N**]	T,N (Transparent,On)
	Rename Object	**bN**,*newname*;	
Line	First Point	**bF,#,#;**	0,0
	Second Point	**bS,#,#;**	0,0
	Rename Object	**bN**,*newname*;	
Polyline	Alignment	**bA**,①,②; ① = [**L**|**M**|**R**] ② = [**T**|**M**|**B**]	L,T (Left,Top)
	Position	**bP**,#,#;	0,0
	Size	**bS**,#,#;	20,20
	Number of Points	**bN**,#;	0
	Offset	**bO**,#x,#y;	0,0
	Point Definition		
	Method 1	**bD**,#s:#e,[**!**]*x0:y0* [**!**]*x1:y1* [**!**]*x2:y2* ... [**!**]*xn:yn*;	
	Method 2 (Linear Fill)	**b**[**X**][**Y**],#s:#e,**F**#b:#i;	
	Method 3	**b**[**X**][**Y**],#s:#e,[**!**]*p0* [**!**]*p1* ... [**!**]*pn*;	

Table C-4. System Parameters.

PARAMETER	FORMAT	DEFAULTS (IF APPLICABLE)
Adjust contrast	**sC,①;** ① = [**D**|**L**|**R**]	
Set contrast	**sC,#;** # = [**0-255**]	135 (240x128); 145 (320x240);
Adjust backlight	**sL,①;** ① = [**O**|**D**|**U**|**N**]	N (On)
Set backlight	**sL,#;** # = [**30-255**]	30 (240x128); 50(320x240);
Set note	**sN,#; # = [0-63]**	58 (D in 8th octave)
Set duration of beep	**sD,#;** # = [**1-65535**]	30 (0.3 seconds)
Beep	**sB;**	
Key beep control	**sK,①,②;** ①= [**O**|**N**] (key down beep) ② = [**O**|**N**] (key up beep)	N (On) O (Off)
Key repeat	**sP,***delay,rate***;** *rate* = 0-1000; *delay* = 0-1000	0,0
Speaker control	**sS,①;** ① = [**O**|**N**]	O (Off)
Return software version	**sV;**	
Return memory available	**sM;**	
Reset the unit	**sR;**	
Define autoexec string	**sA,***$***;** $ = [VTP commands]	sB;sV;
Query error buffer Empty error buffer	**sE;** **sE**,**F;**	
Return display capabilities	**sY;**	
Set timer rate	**sT,#;** # = [**1-1000**]	100 (1 second)
Touch screen enable	**sP,①;** ① = [**O**|**N**]	N (On)
Query system object	**sQ;**	
Enable/disable XON/XOFF and hardware flow control	**sX,①;** ① = [**O**|**N**|**H**]	O (Off)
Write terminal specific settings	**sW;**	
Perform touch screen calibration	**sU,①;** ① = [**O**|**N**]	
Enable/disable power-on setup	**sJ,①;** ① = [**E**|**R**|**D**]	E (Enabled)

Table C-5. VT52 Object Command Summary.

Command Set	Command	Format	Defaults (If Applicable)
VTP	Create VT52 Object	**AV,***name***;**	
	Alignment	**bA,**①**,**②**;** ① = [**L**|**M**|**R**] ② = [**T**|**M**|**B**]	L,T (Left,Top)
	Auto Line Feed	**bL,**①**;** ① = [**O**|**N**]	O (Off)
	Auto Scroll	**bC,**①**;** ① = [**O**|**N**]	O (Off)
	Auto Wrap	**bW,**①**;** ① = [**O**|**N**]	O (Off)
	Font	**bF,***fontname***;**	DF (5x7Font)
	Position	**bP,***#,#***;**	0,0
	Border	**bR,***mask;*	2
	Size	**bS,***#,#***;**	No default. Size must be set before the object can be enabled.
	Tab Stops	**bT,**#:#:#:...;	No default tab stops
	VT52 Emulation Enable	**bE;**	Disabled
VT52 Escape Sequences	Cursor Up	**ESC A**	
	Cursor Down	**ESC B**	
	Cursor Right	**ESC C**	
	Cursor Left	**ESC D**	
	Clear Screen	**ESC E**	
	Set Text Mode	**ESC F** Mode Mode = [**N**|**I**|**F**|**B**]	N (Normal)
	Cursor Home	**ESC H**	
	Alt. Set Cursor Position	**ESC I** # * # = row+64 * = col+64	
	Erase to End of Screen	**ESC J**	
	Erase to End of Line	**ESC K**	
	Query Character	**ESC W**	
	Query Cursor Position	**ESC X**	
	Set Cursor Position	**ESC Y** # * # = row+32 * = col+32	
	Set Cursor Mode	**ESC b** Mode Mode = [**N**|**U**]	U (Underline)
	Exit VT52 Mode	**ESC <**	
VT52 Control Characters	Bell	**0x07**	
	Backspace	**0x08**	
	Tab	**0x09**	
	Linefeed	**0x0A**	
	Vertical Tab	**0x0B**	
	Formfeed	**0x0C**	
	Carriage Return	**0x0D**	
	Delete	**0x7F**	

APPENDIX D.

EXAMPLE VTP COMMANDS

The first step to displaying text is to define a text object. Suppose we want a text object at the top center of the display.

AT,MyField;	define and name the object.

We now have a text field called “My Field” in the default form which is called Main. The following commands set attributes for the text object.

bP,120,0;	Set the *position* of the field to the top-center of the display (120 is the horizontal center, 0 is at the top).
bA,M,T;	Align the middle of the field horizontally at the *position*, and the top of the field vertically at the *position*.
bD,This is text;	Stores a string in the text field.
e;	This command will (finally) cause the stored text string to appear.

To change the text, just send another parameter change command:

bD,NewText;	The old text is deleted and replaced with “New Text”.

To cause the text to flash, just change the attribute parameter so that Flash in ON:

bT,F,N;	“New Text” will now be flashing on the display.

Now create and display a new text object with the following commands:

```
AT,another text field;
bD,This is another text field;
bP,120,50;
bA,M,M;
e;
```

This displays more text on the screen. Perhaps we would like to change the first text object, so that the text is not flashing. First select the object we want, then change the attribute:

```
a,My Field;
bT,F,O;
```

Suppose we want to change the second text object to inverse text (black characters on a white background).

```
a,another text field;
bT,I,N;
```

Now let’s draw a line across the middle of the screen.

```
AL,centerline;
bF,0,64;
bS,239,64;
e;
```

Perhaps we want to erase our first text field from the screen.

```
a,My Field;
d;
```

The text object “My Field” has disappeared from the screen. It still resides in memory and can be redrawn to the screen by enabling it again.

If we are tired of the object, we can delete it altogether:

```
B;
```

Enough fun. Let’s discard all objects from RAM and restore the default objects.

```
IF;
```

The following series of VTP commands creates a form called “BUTTONS” which labels each key with a number. Key objects send the corresponding number to the host when a key is pressed.

```
AF,BUTTONS;

AT,Label00;
bP,20,5;
bA,M,M;
bD,1;
e;

AT,Label01;
bP,60,5;
bA,M,M;
bD,2;
e;

AT,Label02;
bP,100,5;
bA,M,M;
bD,3;
e;

AT,Label03;
bP,140,5;
bA,M,M;
bD,4;
e;

AT,Label04;
bP,180,5;
bA,M,M;
bD,5;
e;

AT,Label05;
bP,220,5;
bA,M,M;
bD,6;
e;

AT,Label10;
bP,20,123;
bA,M,M;
bD,7;
e;

AT,Label11;
bP,60,123;
bA,M,M;
bD,8;
e;

AT,Label12;
bP,100,123;
bA,M,M;
bD,9;
e;

AT,Label13;
bP,140,123;
bA,M,M;
bD,10;
e;

AT,Label14;
bP,180,123;
bA,M,M;
bD,11;
e;

AT,Label15;
bP,220,123;
bA,M,M;
bD,12;
e;

AL,boxtop;
bF,1,12;
bS,238,12;
e;

AJ,KeyLabel00;
bP,20,5;
bA,M,M;
bS,40,10;
bD,1;
e;

AJ,KeyLabel01;
bP,60,5;
bA,M,M;
bS,40,10;
bD,2;
e;

AJ,KeyLabel02;
bP,100,5;
bA,M,M;
bS,40,10;
bD,3;
e;

AJ,KeyLabel03;
bP,140,5;
bA,M,M;
bS,40,10;
```

```
bD,4;
e;

AJ,KeyLabel04;
bP,180,5;
bA,M,M;
bS,40,10;
bD,5;
e;

AJ,KeyLabel05;
bP,220,5;
bA,M,M;
bS,40,10;
bD,6;
e;

AJ,KeyLabel10;
bP,20,123;
bA,M,M;
bS,40,10;
bD,7;
e;

AJ,KeyLabel11;
bP,60,123;
bA,M,M;
bS,40,10;
bD,8;
e;

AJ,KeyLabel12;
bP,100,123;
bA,M,M;
bS,40,10;
bD,9;
e; AJ,KeyLabel13;

bP,140,123;
bA,M,M;
bS,40,10;
bD,10;
e;

AJ,KeyLabel14;
bP,180,123;
bA,M,M;
bS,40,10;
bD,11;
e;

AJ,KeyLabel15;
bP,220,123;
bA,M,M;
bS,40,10;
bD,12;
e;
```

APPENDIX E.

BINARY DATA COMMUNICATION

The QTERM-K65 accepts binary data to define the pixel patterns for bitmap and font objects. The terminal firmware can also be upgraded by sending binary data. This appendix describes the requirements for sending binary data to the K65.

E.1 Binary Data Definitions

The bB command is provided for sending binary data definitions through the serial interface. The command format is as follows:

bB,<#bytes><~#bytes><bindata><CRC>;

where:

<#bytes> =	A 4-byte header indicating the number of bytes in the <bindata> section. The header is binary, MSB (most significant byte) first.
<~#bytes> =	A 4-byte header section containing the complement of <#bytes>, such that the sum of quantities <#bytes> and <~#bytes> equals zero.
<bindata> =	The binary data definition for the object.
<CRC> =	A 2-byte cyclic redundancy check (CRC).

The header indicating the length of the binary data is transmitted after the comma. The binary data are then transmitted to the terminal one byte at a time. Finally, a 16-bit CRC is transmitted, followed by the terminating semicolon. The "<" and ">" characters are shown to clearly separate each token in the header and are not transmitted to the host.

The CRC is used to verify data integrity after transmission. The theory describing CRCs is complex and is not described here. However, CRCs are easy to calculate. Source code for calculating a 16-bit CRC using the C programming language is provided in Table E-1. References with further information on CRCs are also provided.

The QTERM-K65 calculates a CRC on all incoming binary data. If the calculated CRC does not match the value sent at the end of transmission, the data definition is aborted and an error is logged in the K65 error buffer.

A method has been implemented to allow the use of XON/XOFF flow control during binary transfers. This involves mapping the XON, XOFF and escape character (0xBF) sequences. These sequences are decoded back into the original bytes by the K65 before the data is used. The escape character is 0xBF. The mappings are as follows:

0x11 (XON) maps to 0xBF 0x91
0x13 (XOFF) maps to 0xBF 0x93
0xBF (escape) maps to 0xBF 0xBF

The substitutions may be made as the bytes are being transmitted to the terminal. These characters may occur in the <#bytes> field, the <bindata> field or the <CRC> field. NOTE: The mappings must take place for all binary transfers, whether XON/XOFF is enabled or disabled, or if hardware flow control is used instead of XON/XOFF.

E.2 Firmware Upgrade

The S command is provided to upgrade the terminal firmware through the serial interface. The command format is as follows:

S,<#bytes><~#bytes><bindata><CRC>;

where:

<#bytes> =	A 4-byte header indicating the number of bytes in the <bindata> section. The header is binary, MSB (most significant byte) first.
<~#bytes> =	A 4-byte header section containing the complement of <#bytes>,such that the sum of quantities <#bytes> and <~#bytes> equals zero.
<bindata> =	The binary image of the new terminal firmware.

<CRC> = A 2-byte cyclic redundancy check (CRC).

Again, the "<" and ">" characters are shown for clarity only and are not transmitted to the host.

To upgrade your terminal firmware, do the following:

1. Power the K65 down and up.

2. Issue the IF; command to clear the RAM of all objects (but the defaults).

3. Issue the G; command to clear the contents of ROM by saving the empty RAM to ROM.

4. Reset the terminal by sending the sR; command.

5. Issue the S command as described above. The entire command will take from several seconds to a minute to transmit, depending on the current baud rate.

6. When the command has been transmitted and verified, the K65 will execute a write cycle to save the new firmware to ROM. This may take up to 10 seconds to complete. During this time, the terminal is running from code stored in RAM. **Do not power down during this time, or the ROM will become corrupted.**

7. If all has gone well, the terminal will automatically reset after the write cycle. The new firm-ware version number will be displayed briefly. The unit should now run normally.

Remember to prefix any XON, XOFF and escape characters as described previously.

Upgrading the firmware destroys any configuration that is currently stored in ROM. All settings are restored to the factory defaults.

E.3 CRC Source Code

The source code in Table E-1 calculates a 16-bit CRC for a collection of bytes. The **calcCRC16** function takes a pointer to the beginning of the bytes and the number of bytes as parameters. The function returns the calculated 16 bit CRC as an unsigned short integer value. The **crc16Tbl** structure contains values that are necessary for the calculation.

E.4 References

Check these sources for more information on CRCs:

1) "Understanding CRCs", Tim Kientzle, *Dr. Dobb's Journal* #264, April 1997, page 103.

2) Kermit Protocol Reference, available via anonymous FTP from kermit.columbia.edu.

Table E-1. Source Code for CRC Calculation.

```
Const unsigned short crc16Tbl[256] =
{
 0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50A5, 0x60C6, 0x70E7,
 0x8108, 0x9129, 0xA14A, 0xB16B, 0xC18C, 0xD1AD, 0xE1CE, 0xF1EF,
 0x1231, 0x0210, 0x3273, 0x2252, 0x52B5, 0x4294, 0x72F7, 0x62D6,
 0x9339, 0x8318, 0xB37B, 0xA35A, 0xD3BD, 0xC39C, 0xF3FF, 0xE3DE,
 0x2462, 0x3443, 0x0420, 0x1401, 0x64E6, 0x74C7, 0x44A4, 0x5485,
 0xA56A, 0xB54B, 0x8528, 0x9509, 0xE5EE, 0xF5CF, 0xC5AC, 0xD58D,
 0x3653, 0x2672, 0x1611, 0x0630, 0x76D7, 0x66F6, 0x5695, 0x46B4,
 0xB75B, 0xA77A, 0x9719, 0x8738, 0xF7DF, 0xE7FE, 0xD79D, 0xC7BC,
 0x48C4, 0x58E5, 0x6886, 0x78A7, 0x0840, 0x1861, 0x2802, 0x3823,
 0xC9CC, 0xD9ED, 0xE98E, 0xF9AF, 0x8948, 0x9969, 0xA90A, 0xB92B,
 0x5AF5, 0x4AD4, 0x7AB7, 0x6A96, 0x1A71, 0x0A50, 0x3A33, 0x2A12,
 0xDBFD, 0xCBDC, 0xFBBF, 0xEB9E, 0x9B79, 0x8B58, 0xBB3B, 0xAB1A,
 0x6CA6, 0x7C87, 0x4CE4, 0x5CC5, 0x2C22, 0x3C03, 0x0C60, 0x1C41,
 0xEDAE, 0xFD8F, 0xCDEC, 0xDDCD, 0xAD2A, 0xBD0B, 0x8D68, 0x9D49,
 0x7E97, 0x6EB6, 0x5ED5, 0x4EF4, 0x3E13, 0x2E32, 0x1E51, 0x0E70,
 0xFF9F, 0xEFBE, 0xDFDD, 0xCFFC, 0xBF1B, 0xAF3A, 0x9F59, 0x8F78,
 0x9188, 0x81A9, 0xB1CA, 0xA1EB, 0xD10C, 0xC12D, 0xF14E, 0xE16F,
 0x1080, 0x00A1, 0x30C2, 0x20E3, 0x5004, 0x4025, 0x7046, 0x6067,
 0x83B9, 0x9398, 0xA3FB, 0xB3DA, 0xC33D, 0xD31C, 0xE37F, 0xF35E,
 0x02B1, 0x1290, 0x22F3, 0x32D2, 0x4235, 0x5214, 0x6277, 0x7256,
 0xB5EA, 0xA5CB, 0x95A8, 0x8589, 0xF56E, 0xE54F, 0xD52C, 0xC50D,
 0x34E2, 0x24C3, 0x14A0, 0x0481, 0x7466, 0x6447, 0x5424, 0x4405,
 0xA7DB, 0xB7FA, 0x8799, 0x97B8, 0xE75F, 0xF77E, 0xC71D, 0xD73C,
 0x26D3, 0x36F2, 0x0691, 0x16B0, 0x6657, 0x7676, 0x4615, 0x5634,
 0xD94C, 0xC96D, 0xF90E, 0xE92F, 0x99C8, 0x89E9, 0xB98A, 0xA9AB,
 0x5844, 0x4865, 0x7806, 0x6827, 0x18C0, 0x08E1, 0x3882, 0x28A3,
 0xCB7D, 0xDB5C, 0xEB3F, 0xFB1E, 0x8BF9, 0x9BD8, 0xABBB, 0xBB9A,
 0x4A75, 0x5A54, 0x6A37, 0x7A16, 0x0AF1, 0x1AD0, 0x2AB3, 0x3A92,
 0xFD2E, 0xED0F, 0xDD6C, 0xCD4D, 0xBDAA, 0xAD8B, 0x9DE8, 0x8DC9,
 0x7C26, 0x6C07, 0x5C64, 0x4C45, 0x3CA2, 0x2C83, 0x1CE0, 0x0CC1,
 0xEF1F, 0xFF3E, 0xCF5D, 0xDF7C, 0xAF9B, 0xBFBA, 0x8FD9, 0x9FF8,
 0x6E17, 0x7E36, 0x4E55, 0x5E74, 0x2E93, 0x3EB2, 0x0ED1, 0x1EF0
};

/*****************************
* calcCRC16
*****************************/
/* Calculates a 16 bit CRC
*  buf points to data (a bunch of bytes)
*  len indicates number of bytes pointed to by buf
*  This should be the CCITT standard CRC */

unsigned short calcCrc16(char *buf,long len)
{
char *p1;
long i;
unsigned short crc16;

crc16 = 0xffff;

for (i=len,p1=buf; i>0; i--)
        crc16 = crc16Tbl[((crc16>>8) ^ (unsigned short)*(p1++)) & 0x00ff] ^ (crc16<<8);
return(crc16);
} /* end calcCrc16 */
```

APPENDIX F.

KEYPAD VTP COMMANDS

Certain configurations of the QTERM-K65 terminal may include a keypad in addition to or in place of the touch-screen. These objects are very similar to the TouchKey objects described in Chapter 3. This section describes the VTP commands for creating and manipulating these objects.

F.1 Key Object Overview

Key objects are used to define the keystring that will be sent to the host through the serial port when a key on the keypad is pressed. Local keystrings which issue commands to the terminal itself (as if the host had sent the commands) are also supported. Both local and host keystrings may be defined for the same Key object. Keys on the current and ROOT form behave identically to TouchKeys as described in Section 3.5.

There are three attributes for Key objects:

Key Press String - This attribute defines the host and/or local keystrings that are activated when the associated key is pressed. Host keystrings are characters strings that are sent to the host through the serial interface when the key is pressed. Local keystrings contain VTP commands which are executed by the terminal when the key is pressed (as if they were sent by the host). For more information on host and local keystrings, see Section 3.4.4.

Key Release String - This is analogous to the Key Press String attribute, except that it is associated with the release of the key associated with the Key object.

Position - This attribute associates the Key object with a physical key on the keypad.

F.2 Key Object VTP Commands

F.2.1 Create Key Object

Syntax: AK,*name*;

where *name* = name of Key Object.

Description: This command is used to create a new Key object on the current form.

The object name can be any valid name (see Section 3.2) that is not already in use on the current form. Attempting to create an object that already exists logs an error in the error buffer and aborts the create command.

Example: AK,k00;

This creates a Key object named “k00"

F.2.2 Key Press String

This command is identical to Key Press String in Section 3.6.4.12.

F.2.3 Key Release String

This command is identical to Key Release String in Section 3.6.4.13.

F.2.4 Key Position

Syntax: bK,*column*,*row*;

where:
column = 0-7
Row = 0-6

Description: This command associates the Key object to a physical key on the keypad. Column 0 is the leftmost column, and Row 0 is the topmost row on the keypad.

Example: bK,3,5;

This sets the position of the current Key object to column 3, row 5.

APPENDIX G.

AUXILIARY SERIAL PORT

Certain configuration of the QTERM-K65 terminal include hardware support for a secondary EIA-232 serial port in addition to the primary port. This appendix describes VTP extensions that support use of the auxiliary serial port.

All characters received from the auxiliary serial port are transmitted (echoed verbatim) out of the primary serial port. The QTERM-K65 does not support any other processing of these received characters. Transmission of characters out of the auxiliary serial port is controlled through VTP commands as described in the following section.

G.1 Auxiliary Serial Port VTP Commands

G.1.1 Set Serial Port Parameters

Syntax: sp1,①,②;

where:
① = b, p, s or d
② = integer or letter, depending on ①

Description: This commands sets the serial port parameters such as baud rate (①=b), parity (①=p), number of stop bits(①=s) and number of data bits (①=d). Legal values for baud rate are 2400, 4800, 9600, 19200 and 57600. Legal values for parity are 'N' (no parity), 'E' (even parity) and 'O' (odd parity). Legal values for stop bits are 1 and 2. Legal values for data bits are 7 and 8. The default values for the auxiliary serial port are 57600 baud, no parity, 8 data bits and 1 stop bit.

These settings may be saved to ROM with the sW; command.

Example: sp1,b,19200;
sp1,p,N;
sp1,d,8;
sp1,s,1;

This sets the auxiliary serial port settings to 19200 baud, no parity, 8 data bits and 1 stop bit.

G.1.2 Send a String to the Auxiliary Serial Port

Syntax: P1,*string*;

where:
string = the sequence of characters to be transmitted out of the auxiliary serial port.

Description: This command causes *string* to be transmitted out of the auxiliary serial port. Strings containing commas and semicolons may be transmitted by escaping the commands and semicolons with a preceding backslash.

Example: P1,Hello \,world!;

This causes the string "Hello, world!" to be transmitted out of the auxiliary serial port.