Programming for the 700 RF Terminal

WORTH data

Table of Contents

Worth Data Inc.
623 Swift St.
Santa Cruz, CA 95060
ph. 800-345-4220 or 831-458-9938
fax 831-458-9964
email wds@barcodehq.com
http://www.barcodehq.com

Feb. 2005

How the system works...

The RF Terminal has a 6x24 LCD screen and up to 99 voice messages which can be activated by the host user program. Messages from the host user program are written to the serial port to which the applicable Base Station is attached. Up to 64 RF Terminals can be controlled by one base station, so the host user program must address the applicable RF Terminal by its ID character. When the host receives a message from the Base Station, it will receive data with the Terminal ID also included (not true for one-way mode).

There is no programming on the RF Terminal itself. All programming is on the host computer. Any language and/or platform that can read/write to a serial port can easily control a network of RF Terminals.

Some users will prefer sending the formatting sequences directly over the serial port using the Low Level Commands. Others will prefer the Windows ActiveX and TCP/IP controls.

This is how the RF Terminal operates:

Messages from the host user program are sent to the Base Station (via the serial port), then from the Base Station to the RF Terminal. The Terminal responds back to the Base with data and its Terminal ID. The data is then transmitted from the Base to the host computer where it is processed and the next command is sent out. Each RF Terminal has a unique Terminal ID, allowing a single Base Station to handle up to 64 Terminals.

Dialog is established when a Terminal **SIGNS ON** to the RF network. The host computer application waits until a Terminal **SIGNS ON**, then begins its processing by sending the first prompt out to the Terminal via the Base Station. If the Terminal does not receive a prompt from the host, it goes into "sleep" mode, "waking up" and checking with the Base periodically to see if it has any messages waiting. This conserves battery power and reduces radio traffic.

We have tried to make it easy for the programmer to communicate with the Base Station; no protocol or hand-shaking is required. This type of communication is fine when the Base is located only a few feet from the serial port it is connected to. If you are locating your Base Station farther away, use shielded, grounded (bare wire Pin 1 touching shield) cable, lower baud rates and possibly, line drivers for very noisy environments.

Before you begin...

Before you begin programming, there are some factors you should take into consideration during the planning process.

- **Plan for system failures**. This includes hardware failures, software failures and operator failures. In order to create an efficient application, you must put some thought into what you will do when different parts of the system fail.

- **Look for All Errors**. Be sure your program is trapping all possible error conditions that the Base Station may return to you. The list includes:

  Sequence Errors detected
  Illegal Command detected
  Base Station Initialized
  Addressing a Terminal Not Signed In

  Forgetting to program for these error conditions is a common mistake. Even though you think your code will never make a mistake, take advantage of the feedback that the Base Station provides.

- **Parse the Returned Strings thoroughly**. Don't assume anything about the next response from the Base to your program and look only for the partial string such as the ID only; parse the returned string completely and be sure you are examining every possibility. Failure to do so is a common mistake.

- **Plan for expansion**. You may start small (1 Base/1 Terminal) but try to create an application that will allow for easy expansion and addition - especially of Terminals.

- **Site Evaluation**. Site Testing does not require that you have an application up and running and can save you time when you do sit down to create your program if you already know what you will be dealing with in terms of Base Stations and Relays.

- **Use the Demo Programs**. The demo programs can at least allow you to see how the system functions and whether you can anticipate any system-wide problems. The demo programs should also be used as a response-time benchmark.

Planning for failures...

Hardware Failures

Let's assume that each part of the system has failed. How are you going to know what has happened and how are you going to recover?

- The most frequent failures are at the Terminal level. If a Terminal has a hardware failure, it will not be able to **SIGN OUT**. It is possible for the Terminal operator to press the ON/OFF key or the F1 key by accident, forcing the Terminal to **SIGN OUT** - sometimes in the middle of a transaction. This happens at battery-changing time also. You need to plan for partial transactions - do you trash the data you do have and start over, or pick up where you left off?

- Keep in mind that if a Terminal has **SIGN**ed **OUT** in mid-transaction, the Base Station clears any pending message for that Terminal before it will allow it to **SIGN ON** again. Make allowances to re-send messages or prompts that were cleared upon **SIGN ON** if necessary.

- Relay Station failures are often cable-related. If a Terminal puts out a "Who Can Hear Me?" message and a Relay that is for some reason not connected to the Base Station (bad cable, cut cable, broken connectors) hears it, it answers with the message:

  Relay n Cannot Be
  Heard by the Base
  Notify Supervisor
  Press Any Key

  At this point, it is up to the operator to notify someone that the Relay is not communicating with the Base and to check the cabling first. There is no message sent to the host, so it is very important that the operator that receives this message notify someone immediately.

Operator Errors

- Plan on your operator walking out of range and going to lunch in the middle of a transaction. What do you do with the data you do have, and where are you going to start up again?

- Let's say your operator is **SIGNED ON** and decides it's time to take a break. Instead of pressing the F1 key to **SIGN OUT**, he presses the OFF key. Pressing the OFF key is OK (it will **SIGN** him **OUT**) but there is a delay until the **SIGN OUT** is acknowledged. Because of the delay, the operator might think he didn't press the key hard enough and press it again - this time actually powering down the Terminal before the **SIGN OUT** was complete. If this happens, you need to plan to resend the last prompt to the Terminal when he **SIGN**s **ON** again.

PromptCOM Active X Control

Drop-in components are tools that are added to your programming environment "tool kit". There are a variety of different technologies around for implementing a drop-in component such as VBX (for Visual Basic) and VCL (for Delphi and C Builder) and COM (ActiveX). Only the **ActiveX** variety are widely compatible with almost all development environments.

PromptCOM/ActiveX is a drop in COM component that allow programmers to easily add the ability to send prompts to and receive data from their R/F Terminal via an RF Base Station. It is compatible with Visual Basic, Visual C++, Delphi, and most other 32-bit development platforms. See the help file for installation instructions.

Programming Considerations

Before making any method calls, make sure to:

- Set the COM port properties (device name, baud, parity, bits, etc.) as desired. Make sure the port is closed (call **CloseDevice**) before making changes to any of the port settings.
- Call the **OpenDevice** method. This activates the COM port used by this instance of the **WDterm** control.
- Set the **ActiveTerminal** property to identify the terminal on which you desire to operate. You can change the **ActiveTerminal** at any time in order to direct commands to appropriate terminals.

Test For Good Communication

- Implement an *event handler* for **OnTermBaseRegister** that causes a beep or displays a message when called. If communication between the host PC and the base station is good, your *event handler* will fire when your program is running and you power up an attached base station.

Multiple Base Stations

- For installations using multiple base stations attached to a single host PC (these were called "channels" in PromptCOM/DLL) simply add a **WDterm** control to your application for each base station.

Terminal Tracking

- Since you get one set of *event handlers* for each base station, you will need some scheme for keeping track of where each terminal (up to 64 per base station) is in its transaction sequence. One possible solution is to use a "*state*"variable for each terminal (perhaps stored in an array). Test the *state* variable to determine the next prompt for any given terminal.

- It is very important to keep track of "*login status*" for each terminal. Every **SignOut** event should have an associated **SignIn** event and a given terminal should not be allowed to **SignIn** twice without an intervening **SignOut**. Multiple **SignIn**s from one terminal without appropriate **SignOut**s indicate either:

  1) A terminal going out of range and having its power cycled before returning within range **OR**

  2) Two (or more) terminals using the same *ID* (*terminal ID* conflict).

Concepts

When you use drop-in components in your program you will follow the standard object-oriented programming paradigm that uses ***properties***, ***methods***, and ***events*** to implement the functionality of the drop-in component.

- ***Properties*** are the various configuration variables used by the drop-in component. An example of a property is the **ComDeviceName** setting.

- ***Methods*** are function calls used to issue commands and access features of the drop-in component. An example of a method is sending an Input command to the terminal.

- ***Events*** are function definitions placed in your application's source code. The function definitions in your source code are called *Event Handlers*. The skeleton structure of the event handler's source code is automatically generated. The code in the *Event Handler* is called ("fired") by the drop-in component when a specific event occurs. An example of an event is when a terminal returns data and the **OnTermData** event is fired.

The details of how to access ***Properties/Methods/Events*** varies between development platforms. Details of how it works in some of the most popular platforms is illustrated in the samples included with the RF Utilities CD or available for download from our website at:

http://www.barcodehq.com/utilities/WDterminal.exe

Properties

Properties are the various configuration variables used by the **WDterm** control. They are directly assignable in your application (e.g. "**WDterm.ActiveTerminal = 5**") and can be set in your development environment's object browser.

Important: Except for **ActiveTerminal** and **Quiet**, all properties require the serial port to be "closed" before they can be changed. Use the **CloseDevice** method before setting properties and then call **OpenDevice** to re-open the serial port.

Note that your development environment may show more properties for the WDterm control than are listed here. This is normal. You may ignore properties you see that are not listed here.

ActiveTerminal

Valid values: *0* through *63.*
Definition: This is the *terminal ID* (0-63) to which method call instructions are directed .

ComDeviceName

Valid values: *COM1-COM16*
Definition: This is the *serial port* that this instance of the control will use. If you have more than one base station, drop in another **WDterm** control and set its **ComDeviceName** for your other COM port(s).

ComBaudValue

Valid values: *300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200.*
Definition: This is the *serial port speed setting* and must match the base station setting.

ComParity

Valid values: *None, Even, Odd.*
Definition: This is a serial port setting and must match the base station setting. **WDterm** may allow other settings but those listed here are the only ones compatible with current version base stations.

ComDataBits

Valid values: *7, 8*
Definition: This is a serial port setting and must match the base station setting. **WDterm** may allow other settings but those listed here are the only ones compatible with current version base stations.

ComStopBits

Valid values: *1, 2.*
Definition: This is a serial port setting and must match the base station setting. **WDterm** may allow other settings but those listed here are the only ones compatible with current version base stations.

Quiet

Valid values: *True, False.*

Definition: If **Quiet** is set to *True* then any status and error message generated by **WDterm** will be suppressed.

Methods

Methods are commands that you issue to the **WDterm** control. All of the "**Inputxxx**" commands cause the terminal to wait for operator input.

Note that your development environment may show more available methods for the WDterm control than are listed here. This is normal. You may ignore methods you see that are not listed here.

Important: When your application starts up, the serial port is "closed". You must call **OpenDevice** before other method calls will work.

Except for the **ReInitAll** method, all methods use the **ActiveTerminal** ***property*** to identify the terminal to use.

OpenDevice

Function: Opens the communications (serial) port. This must be called before any of the methods described below. Make sure to set all ***Properties*** as desired before calling this method (except **ActiveTerminal** or **Quiet**).

CloseDevice

Function: Closes the communications (serial) port. This must be called before changing any of the ***Property*** settings (except **ActiveTerminal** or **Quiet**). When your application starts up, the serial port is "closed". You must call **OpenDevice** before other method calls will work.

InputAny

Parameters: *line, position, prompt, shifted, timestamped*

Function: This instructs the **ActiveTerminal** to display the *prompt* at *line* and *position* and wait for data to be entered from either terminal keypad or scanner. If *shifted* is set to "true", the terminal will start in shifted mode. *Timestamped* appends a (hhmmss) prefix to the returned data.

InputKeyBd

Parameters: *line, position, prompt, shifted, timestamped*

Function: This instructs the **ActiveTerminal** to display the *prompt* at *line* and *position* and wait for data to be entered from the terminal keypad only. If *shifted* is set to “true”, the terminal will start in shifted mode. *Timestamped* appends a (hhmmss) prefix to the returned data.

InputScanner

Parameters: *line, position, prompt, allowbreakout, timestamped*

Function: This instructs the **ActiveTerminal** to display the *prompt* at *line* and *position* and wait for data to be entered from the terminal scanner only. Setting *allowbreakout* to true allow user to "break out" of scanner only mode by pressing the end key on the terminal. A ***termID+CR*** will be sent to the host.

InputYesNo

Parameters: *line, position, prompt*

Function: This instructs the **ActiveTerminal** to display the *prompt* at *line* and *position* and wait for a ***Yes*** (Enter key or C key) or a ***No*** (0 key or B key) from the terminal keypad.

Note: C and B keys are used to facilitate keypad entry while scanning with the integrated laser.

InputPassword

Parameters: *line, position, prompt, shifted*

Function: This instructs the **ActiveTerminal** to display the *prompt* at *line* and *position* and wait for data to be entered from the terminal keypad only. The entered data is **not** displayed on the terminal.

InputSerial

Parameters: *line, position, prompt*

Function: This instructs the **ActiveTerminal** to display the *prompt* at *line* and *position* and wait for data to be received through the terminal serial port. Waiting for serial input can be bypassed by pressing the **enter** key on the terminal which will send an empty data string to the host (fires the **OnTermData** event handler).

OutputSerial

Parameters: *data*

Function: This instructs the **ActiveTerminal** to send *data* to the terminal's serial port. Data must be less than 231 characters in length for each call to **OutputSerial**. If you are sending data to a printer other than the Cameo or QL3 attached to the terminal, you may need to set the Protocol parameter in the RF Terminal to XON/XOFF. See the RF Terminal Manual for details.

SendDisplay

Parameters: *line, position, prompt*

Function: This instructs the **ActiveTerminal** to display the *prompt* at *line* and *position*. Must be followed by an "**Input**" ***method*** call to take effect.

ClearScreen

Function: This instructs the **ActiveTerminal** to clear its display. Must be followed by an "**Input**" method call to take effect.

ClearLine

Parameters: *line*

Function: This instructs the **ActiveTerminal** to clear the specified *line* on its display. Must be followed by an "**Input**" ***method*** call to take effect.

SendDate

Parameters: *line*

Function: This instructs the **ActiveTerminal** to display date and time on

the specified *line* number. Must be followed by an "**Input**" ***method*** call to take effect.

Beep

Parameters: *count*

Function: This instructs the **ActiveTerminal** to beep *count* times. *Count* may be a value from 1 to 9. Must be followed by an "**Input**" ***method*** call to take effect.

PlayVoice

Parameters: *msgnum*

Function: This instructs the **ActiveTerminal** to play voice message number *msgnum*. *Msgnum* may be a value from 1 to 99. Must be followed by an "**Input**" ***method*** call to take effect.

ReInit

Function: This instructs the **ActiveTerminal** to re-initialize. Must be followed by an "**Input**" ***method*** call to take effect.

NOTE: *Base Stations use the message "Buffer Reinitialized..." to indicate a single terminal re-initialization.*

ReInitAll

Function: Instructs all attached terminals to re-initialize.

Events

WDterm ***events*** occur when a specific condition is met. When an ***event*** is "fired", an ***event handler*** function in your application is called.

Though the details of exactly how it is done varies from one programming environment to the next, the source code skeletons for the various ***event handlers*** are automatically generated and inserted into your source code for you. See the samples for more specific information.

Each ***event*** passes relevant information to your ***event handler*** function. The only event that does not pass any data is **OnTermBaseRegister**. All others pass at least the ***Terminal ID*** on which the event occurred. **OnTermData** also passes the ***data*** that was keyed or scanned into the terminal.

Terminal ID is always passed as 0-63. A ***Terminal ID*** value of 99 indicates an error.

Once you have the ***event handler*** skeletons, you can proceed to add whatever functionality you desire to each event.

You must call the **OpenDevice** ***method*** before any ***events*** can be fired.

OnTermBaseRegister

Event: An attached base station has successfully powered up and communicated with the host computer via the serial connection.

OnTermSignIn6

Data passed: *terminal*
Event: A six-line terminal has signed in. Terminal ID is passed in *terminal*.

OnTermSignIn4

Data passed: *terminal*
Event: A four-line terminal has signed in. Terminal ID is passed in *terminal*.

OnTermSignOut

Data passed: *terminal*
Event: A terminal has signed out. Terminal ID is passed in *terminal*.

OnTermData

Data passed: *terminal, data*
Event: A *terminal* has sent *data* in response to an **Input** ***method*** call.

OnTermNotSignedIn

Data passed: *terminal*
Event: A command has been sent to a *terminal* that is not signed in.

OnTermSequenceError

Data passed: *terminal*
Event: The one-for-one host prompt/*terminal* response protocol has

been violated. The host cannot send a second **Input** command until it has received a response from the first **Input** command. If a base station receives 5 sequence errors in a row, a ***Host Logic*** error is generated and the base shuts itself down.

While PromptCom/ActiveX will intercept and prevent most logic errors, they are still possible so you should implement this event handler!

OnTermIllegalCommand

Data passed: *terminal*
Event: An illegal command has been sent to a *terminal*.

PromptCom/ActiveX is designed to prevent illegal commands but software is not always perfect and we may not have imagined all the ways in which our customers will want to use it!

OnTermUpArrow

Data passed: *terminal*
Event: The up-arrow button has been pressed on a *terminal*. You must issue another **Input** ***method*** call before **WDterm** can respond to another keypress on the terminal.

OnTermDownArrow

Data passed: *terminal*
Event: The down-arrow button has been pressed on a *terminal*. You must issue another **Input** ***method*** call before **WDterm** can respond to another keypress on the terminal.

OnTermLeftArrow

Data passed: *terminal*
Event: The left-arrow button has been pressed on a *terminal*. You must issue another **Input** ***method*** call before **WDterm** can respond to another keypress on the terminal.

OnTermRightArrow

Data passed: *terminal*
Event: The right-arrow button has been pressed on a *terminal*. You must issue another **Input** ***method*** call before **WDterm** can respond to another keypress on the terminal.

OnTermBeginKey

Data passed: *terminal*
Event: The BEGIN button has been pressed on a *terminal*. You must issue another **Input** ***method*** call before **WDterm** can respond to another keypress on the terminal.

OnTermEndKey

Data Passed: *terminal*
Event: The END button has been pressed on a *terminal*. You must issue

another **Input** method call before **WDterm** can respond to another keypress on the terminal.

OnTermSearchKey

Data passed: *terminal*

Event: The SEARCH button has been pressed on a *terminal*. You must issue another **Input** method call before **WDterm** can respond to another keypress on the terminal.

PromptNET TCP/IP Active X Control

PromptNET/ActiveX is a drop in COM component that allow programmers to easily add the ability to send prompts to and receive data from their R/F Terminal via an RF Base Station across a TCP/IP network connection.

PromptNET requires a "Client" computer on a TCP/IP network (to which up to 4 serial Base Stations can be attached) and a "Server" computer visible on the network to the Client.

The client computer runs the **PromptNET Client Utility** program as a background task. The server computer runs your application which uses the **PromptNET** ActiveX component to communicate with the Client.

The ActiveX component is compatible with Visual Basic, Visual C++, Delphi, and most other 32-bit development platforms. The Client Utility program requires Windows 95 or later, the Server ActiveX control requires Windows 98 or later. See the help file for installation instructions.

Programming Considerations

Network Setup

- The network settings on both client and server must support TCP/IP communications.
- It is critical that the client and server computers are "visible" to each other across your network. Both computers must have an IP address in the same subnet. The server requires a static IP address while the Client can either have a static address or use an assigned IP address via a DHCP server or equivalent. Refer to your Windows networking administration utility in the Control Panel to configure computer IP address settings.
- PromptNET uses ports 54123 (server) and 54124 (client).
- You can link server and client through a dial-up or DSL internet link as long as the server has a static IP address and your router passes the above ports.
- If you are unsure of how to set up your IP configuration properly, refer to your network administrator for help.

Client Utility

- Make sure the Client Utility is properly installed on the client computer and communicating with at least one Base Station. Test the Client by cycling power on the Base Station. You should see a "Base SignOn" message in the monitor window.

Server Communications

- Run the ***Server Test Utility*** on the server computer. Go to the client computer, set the IP address for the server computer and a unique "**Base Name**" for the ***Client Utility*** and attempt to connect to the ***Server Test Utility***. If the ***Client Utility*** connects, you are configured properly. Go to the server computer, shut down the ***Server Test Utility*** and begin work on your **PromptNET** server application.

- For Client/Server communications, the ***Client Utility*** is required to be running on the PC that the serial Base Stations are attached to.

- Before making any **WDIPterm** method calls in your application, make sure to set the **ServerOn** property to "true".

Test For Good Communication

- Implement an ***event handler*** for **OnTermBaseRegister** that causes a beep or displays a message when called. If communication between the host PC and the base station is good, your ***event handler*** will fire when your program is running and you power up an attached base station.

Multiple Base Stations

- For installations using multiple base stations attached to a single client PC, simply use the four "channels" provided by the ***Client Utility*** program.

Multiple Client Computers

- The number of client computers is limited only by the processing capability of your server.

Terminal Tracking

- Since you get only one set of ***event handlers***, you will need some scheme for keeping track of where each terminal (up to 64 per base station, up to 4 base stations per client) is in its transaction sequence. One possible solution is to use a "state" variable for each terminal (perhaps stored in an array). Test the state variable to determine the next prompt for any given terminal. See the samples on the RF Terminal Utilities CD for more ideas (You can also download them from our website:
  http://www.barcodehq.com/utilities/PromptNETActiveX.exe)

- It is very important to keep track of "login status" for each terminal. Every **SignOut** ***event*** should have an associated **SignIn** ***event*** and a given terminal should not be allowed to **SignIn** twice without and an intervening **SignOut**. Multiple **SignIns** from one terminal without appropriate **SignOut**s indicate either:

  1) A terminal going out of range and having its power cycled before returning within range OR

  2) Two (or more) terminals using the same *ID* (*terminal ID* conflict).

Concepts

Drop-in components are tools that are added to your programming environment "tool kit". Only the ActiveX variety are widely compatible with almost all development environments. When you use drop-in components in your program you will follow the standard object-oriented programming paradigm that uses ***properties***, ***methods***, and ***events*** to implement the functionality of the drop-in component.

Properties are the various configuration variables used by the drop-in component. An example of a property is the **ServerOn** setting.

Methods are function calls used to issue commands and access features of the drop-in component. An example of a method is sending an **Input** command to the terminal.

Events are function definitions placed in your application's source code. The function definitions in your source code are called *Event Handlers*. The skeleton structure of the event handler's source code is automatically generated. The code in the *Event Handler* is called ("fired") by the drop-in component when a specific event occurs. An example of an event is when a terminal returns data and the **OnTermData** event is fired.

The details of how to access ***Properties/Methods/Events*** varies between development platforms. Details of how it works in some of the most popular platforms is illustrated in the samples included with the RF Utilities CD or available for download from our website at:

http://www.barcodehq.com/utilities/PromptNETActiveX.exe

Properties

Properties are the various configuration variables used by the **WDIPterm** control. They are directly assignable in your application (eg. "**WDIPterm.ServerOn = true**") and can be set in your development environment's object browser.

Note that your development environment may show more properties for the **WDIPterm** *control than are listed here. This is normal. You may ignore properties you see listed in your development environment that are not listed here.*

ServerOn

Valid values: *True, False*.
Function: Set to *True* to enable the server. Set to *False* to turn the server off. You should leave this off unless your program is actually running. Setting it to *True* at design-time can cause problems.

Quiet

Valid values: *True, False*.
Function: If **Quiet** is set to *True* then any status and error message generated by **WDIPterm** will be suppressed.

LogFile

Valid values: *blank* or a valid *file name*.
Function: If the *file* does not exist it will be created. If it exists, it will be appended to. If **LogFile** is blank, no log file is maintained.

LogFileSize

Valid values: *1000 - 10000000*
Function: Sets maximum log file size. When file size is exceeded, oldest entries are removed first.

ClientList

Valid Values: Read Only.
Function: Returns a formatted string listing all attached client **BaseNames** and associated IP numbers.
Format is "**basename CR/LF ip address CR/LF basename…**".

Methods

Methods are commands that you issue to the **WDIPterm** control. All of the "**Inputxxx**" commands cause the terminal to wait for operator input.

Note that your development environment may show more available methods for the **WDterm** *control than are listed here. This is normal. You may ignore methods you see that are not listed here.*

InputAny

Parameters: *basename, channel, terminal, line, position, prompt, shifted, timestamped*

Function: This instructs the *terminal* attached to client *basename* on *channel* to display the *prompt* at *line* and *position* and wait for data to be entered from either terminal keypad or scanner. If *shifted* is set to "**true**", the terminal will start in shifted mode. *Timestamped* appends a (hhmmss) prefix to the returned data.

InputKeyBd

Parameters: *basename, channel, terminal, line, position, prompt, shifted, timestamped*

Function: This instructs the *terminal* attached to client *basename* on *channel* to display the *prompt* at *line* and *position* and wait for data to be entered from the terminal keypad only. If *shifted* is set to "**true**", the terminal will start in shifted mode. *Timestamped* appends a (hhmmss) prefix to the returned data..

InputScanner

Parameters: *basename, channel, terminal, line, position, prompt, allowbreakout, timestamped*

Function: This instructs the *terminal* attached to client *basename* on *channel* to display the *prompt* at *line* and *position* and wait for data to be entered from the terminal scanner only. Setting allow breakout to true allow user to "breakout" of scanner only mode by pressing the end key on the terminal. A **termID+CR** will be sent to the host.

InputYesNo

Parameters: *basename, channel, terminal, line, position, prompt*

Function: This instructs the *terminal* attached to client *basename* on *channel* to display the *prompt* at *line* and *position* and wait for a **Yes** (Enter key or C key) or a **No** (0 key or B key) from the terminal keypad.

Note: C and B keys are used to facilitate keypad entry while scanning with the integrated laser.

InputPassword

Parameters: *basename, channel, terminal, line, position, prompt, shifted*
Function: This instructs the *terminal* attached to client *basename* on *channel* to display the *prompt* at *line* and *position* and wait for data to be entered from the terminal keypad only. The entered data is not displayed on the terminal.

InputSerial

Parameters: *basename, channel, terminal, line, position, prompt*
Function: This instructs the *terminal* attached to client *basename* on *channel* to display the *prompt* at *line* and *position* and wait for data to be received through the terminal serial port. Waiting for serial input can be bypassed by pressing the **enter key** on the terminal which will send an empty data string to the host (fires the **OnTermData** event handler).

OutputSerial

Parameters: *basename, channel, terminal, data*
Function: This instructs the *terminal* attached to client *basename* on *channel* to send *data* to the terminal's serial port. Data must be less than 231 characters in length for each call to **OutputSerial**. If you are sending data to a printer (other than the Cameo or the QL3) attached to the terminal, you may need to set the **Protocol** parameter in the RF Terminal to XON/XOFF. See the RF Terminal Manual for details.

SendDisplay

Parameters: *basename, channel, terminal, line, position, prompt*
Function: This instructs the *terminal* attached to client *basename* on *channel* to display the *prompt* at *line* and *position*. Must be followed by an "**Input**" ***method*** call to take effect.

ClearScreen

Parameters: *basename, channel, terminal*
Function: This instructs the *terminal* attached to *client basename* on *channel* to clear its display. Must be followed by an "**Input**" ***method*** call to take effect.

ClearLine

Parameters: *basename, channel, terminal, line*
Function: This instructs the *terminal* attached to client *basename* on *channel* to clear the specified *line* on its display. Must be followed by an "**Input**" ***method*** call to take effect.

SendDate

Parameters: *basename, channel, terminal, line*

Function: This instructs the *terminal* attached to client *basename* on *channel* to display date and time on the specified *line* number. Must be followed by an "**Input**" ***method*** call to take effect.

Beep

Parameters: *basename, channel, terminal, count*

Function: This instructs the *terminal* attached to client *basename* on *channel* to beep *count* times. *Count* may be a value from 1 to 9. Must be followed by an "**Input**" method call to take effect.

PlayVoice

Parameters: *basename, channel, terminal, msgnum*

Function: This instructs the *terminal* attached to client *basename* on *channel* to play voice message number *msgnum*. *Msgnum* may be a value from 1 to 99. Must be followed by an "**Input**" ***method*** call to take effect.

ReInit

Parameters: *basename, channel, terminal*

Function: This instructs the *terminal* attached to client *basename* on *channel* to re-initialize. Must be followed by an "**Input**" ***method*** call to take effect.

*NOTE: Base Stations use the message "**Buffer Reinitialized...**" to indicate a single terminal re-initialization.*

ReInitAll

Parameters: *basename, channel*

Function: This instructs all *terminals* attached to client *basename* on *channel* to re-initialize.

TestClient

Parameters: none

Function: This instructs the Server to "ping" all attached clients. Results are recorded in the log.

PageClient

Parameters: *basename, message, beeps*

Function: Sends a *message* and/or *beeps* to the client computer *basename*.

Events

WDIPterm *events* occur when a specific condition is met. When an ***event*** is "fired", an *event handler* function in your application is called.

Though the details of exactly how it is done varies from one programming environment to the next, the source code skeletons for the various *event handlers* are automatically generated and inserted into your source code for you. See the samples for more specific information.

Each ***event*** passes relevant information to your *event handler* function. **OnTermData** passes the data that was keyed or scanned into the terminal.

Terminal ID is always passed as 0-63. A terminal ID value of 99 is used as a placeholder for logging purposes.

Once you have the *event handler* skeletons, you can proceed to add whatever functionality you desire to each ***event***.

You must set the **ServerOn** property to **true** before any ***events*** can be fired.

OnTermBaseRegister

Data passed: *basename, channel*
Event: An attached base station on client *basename* has successfully powered up on *channel* and communicated with the host computer via the serial connection.

OnTermSignIn6

Data passed: *basename, channel, terminal*
Event: A six-line *terminal* has signed in on *channel* at client *basename*. Terminal ID is passed in *terminal*.

OnTermSignIn4

Data passed: *basename, channel, terminal*
Event: A four-line *terminal* has signed in on *channel* at client *basename*. Terminal ID is passed in *terminal*.

OnTermSignOut

Data passed: *basename, channel, terminal*
Event: A *terminal* has signed out on *channel* at client *basename*. Terminal ID is passed in *terminal*.

OnTermData

Data passed: *basename, channel, terminal, data*
Event: A *terminal* on *channel* at client *basename* has sent *data* in response to an **Input *method*** call.

OnTermNotSignedIn

Data passed: *basename, channel, terminal*
Event: A command has been sent to a terminal that is not signed in.

OnTermSequenceError

Data passed: *basename, channel, terminal*
Event: The one-for-one host prompt/terminal response protocol has been violated. The host cannot send a second **Input** command until it has received a response from the first **Input** command. If a base station receives 5 sequence errors in a row, a Host Logic error is generated and the base shuts itself down.

While PromptNET/ActiveX will intercept and prevent most logic errors, they are still possible so you should implement this event handler!

OnTermIllegalCommand

Data passed: *basename, channel, terminal*
Event: An illegal command has been sent to a terminal.

PromptNET/ActiveX is designed to prevent illegal commands but software is not always perfect and we may not have imagined all the ways in which our customers will want to use it!

OnTermUpArrow

Data passed: *basename, channel, terminal*
Event: The up-arrow button has been pressed on a *terminal*. You must issue another **Input** method call before **WDIPterm** can respond to another keypress on this terminal.

OnTermDownArrow

Data passed: *basename, channel, terminal*
Event: The down-arrow button has been pressed on a *terminal*. You must issue another **Input** method call before **WDIPterm** can respond to another keypress on this *terminal*.

OnTermLeftArrow

Data passed: *basename, channel, terminal*
Event: The left-arrow button has been pressed on a *terminal*. You must issue another **Input** method call before **WDIPterm** can respond to another keypress on this *terminal*.

OnTermRightArrow

Data passed: *basename, channel, terminal*
Event: The right-arrow button has been pressed on a *terminal*. You must issue another **Input** method call before **WDIPterm** can respond to another keypress on this *terminal*.

OnTermBeginKey

Data passed: *basename, channel, terminal*

Event: The BEGIN button has been pressed on a *terminal*. You must issue another **Input** method call before **WDIPterm** can respond to another keypress on this *terminal*.

OnTermEndKey

Data passed: *basename, channel, terminal*

Event: The END button has been pressed on a *terminal*. You must issue another **Input** method call before **WDIPterm** can respond to another keypress on this *terminal*.

OnTermSearchKey

Data passed: *basename, channel, terminal*

Event: The SEARCH button has been pressed on a *terminal*. You must issue another **Input** method call before **WDIPterm** can respond to another keypress on this *terminal*.

PromptCOM DLL for Windows

The **PromptCOM Dynamic Link Library** (**DLL**) is for use with Windows programming environments that cannot use the ActiveX components. We highly recommend using the ActiveX components if you can.

You can find the program on the RF Terminal Utilities CD or download the file from our website at **http://www.barcodehq.com/utilities/winterm.exe**. To install the program, run the INSTALL.EXE program from Windows Explorer. The program, PromptCOM comes in both 16 bit and 32 bit versions of a Windows Dynamic Link Library (DLL) that allows programmers to easily add the ability to send prompts and receive data from their RF Terminal via the RF Base Station or direct serial link.

The Application Programming Interface (API) for **PromptCOM** consists fo the following functions:

InitComDLL . . . . . .Initializes the **PromptCOM** system
CloseComDLL . . . . . .Shuts down the **PromptCOM** system and frees resources without closing the parent application.
Setup . . . . . .Setup is used to configure the COM port.
SendCommand . . . . . .This function sends a command to the terminal with the given ID.
GetCommData . . . . . .This function returns the data entered at the remote unit for that prompt.
DataAvailable . . . . . .Use this function to check if there is data to process before calling **GetCommData**.

There are sample programs distributed on the Utilities CD for Visual Basic, Access and Delphi. There is also a Visual Basic code example that does not require the DLL. Use the Help System as documentation and view the README file for the latest changes.

Low Level Programming Commands

The low-level programming commands are provided here in case you cannot for some reason use the ActiveX controls (i.e. programming in DOS, UNIX, etc..).

Host to Terminal Programming

The basic format of a message transmitted from Host to Base to Terminal is simple:

Byte position	Function	Possible values
1	**RF Terminal ID**	***0-9, A-Z, a-z, - =***
2+	**Command(s)**	**
Last	**Termination of message**	***EOT*** (ASCII 4)

The **RF Terminal ID** is always the first byte and always only 1 character in length. There are 64 different possible values - 0-9, A-Z, a-z, - and =.

The **Command(s)** section of the message always starts with the second byte and can consist of one or more commands - including data to be displayed or voice messages to be broadcast.

The last byte is always ASCII 4 (EOT) to **terminate** the message.

Here is a listing of valid commands and examples:

Command characters	Command function

***@**.....Re-initializes all terminals
3@.....Re-initializes Terminal #3
1@Bn.....Make Terminal #1 beep n (1-9) times
2@C0* .....Clears the entire screen (4 lines or 6 lines) on Terminal #2.
0@C1.....Clears line 1 on Terminal #0
1@C2.....Clears line 2 on Terminal #1
2@C3.....Clears line 3 on Terminal #2
0@C4.....Clears line 4 on Terminal #0
3@C5* .....Clears line 5 on Terminal#3 (if 6 line display), Clears all if 4 line
1@C6* .....Clears line 6 on a 6 line display. Will do nothing on a 4 line.
1@Dn.....Displays date and time on line n (1-4) in US (mm/dd/yy, hh:mm:ss) or Euro (dd/mm/yy, hh:mm:ss) format on Terminal #1
1@Vnn.....Play voice message #nn (01-99) on Terminal #1
1@Sdataxxxx....Output dataxxxxxxx to serial port on Terminal #1 -max 231 chs

A typical "prompt" command sequence follows the format below:

0@n,m,o,data

where **n**.....is the line number (1-4) you want the prompt displayed on
m.....is the character position (1-20) where you want the prompt displayed
o.....is the character that determines whether the prompt is for display only (0) or is waiting for data input (1) See the table below for valid characters for this position.
data...is the data you want displayed

For example, the command **@1,1,1, Enter Quantity** would display "*Enter Quantity*" starting at position 1 on line 1, then wait for the operator to enter data.

These are valid entries for the third position character:

0.....No data input for this Command, Display ONLY
1.....Data input required from the keypad or scanner
2.....Only keypad input allowed, start un-shifted
3.....Only keypad input allowed, start SHIFTED
4.....Only scanner input allowed
5.....Only accept YES (Enter key) or NO (0 key) keypad response. (Terminal sends 1 for YES, 0 for NO)
A.....same as 1, but time stamped as prefix (hhmmss)
B.....same as 2, but time stamped as prefix (hhmmss)
C.....same as 3, but time stamped
D.....same as 4, but time stamped
E.....same as 4, but can press END key to break-out of scanner-only input mode. Terminal ID + CR is sent to host
S.....SHIFTED keypad input or scanner input
p.....un-shifted keypad entry with no display (for passwords)
P.....SHIFTED keypad entry with no display (for passwords)
R.....Data input required from the RS-232 serial port (waiting for serial input can be bypassed by pressing the ENTER key which will send a NULL data string back to host computer.)Uses for this are PDF 417 Serial Scanners, and the Cameo Printer's magnetic stripe input. A POS terminal becomes possible. Scan the credit card and print the receipt, all on the RF Terminal.
K.....Data input from an external serial keyboard that attaches to the serial port. As data is keyed, the characters are displayed on the RF Terminal LCD display.

Here are some rules and useful tips for creating commands:

- Re-initialize commands ***@**, or **x@** clear the buffer for terminals in the Base Station. Following a reinitialization, the host program should re-display of all the screen data necessary to start the application.

- A message with multiple commands is legal and useful. For example, the command "**@1,1,0, PLEASE ENTER@2,1,1,QTY**" would display PLEASE

ENTER on line 1, display QTY on line 2, then wait for data input. All 6 lines can be filled with one message.

- Messages can be a combination of multiple commands, (i.e. voice messages, initialization, clearing lines, requesting data entry), up to 231 characters in length. A message cannot though, contain an **@S** command in combination with any other command. A message also should not contain more than 1 request for data entry (third character in command is 1). For example:

  @1,1,1,ITEM@2,1,1,QTY

  has two data entry "prompt" commands combined. If this message were sent to the RF Terminal, the first data entry prompt (**@1,1,1,ITEM**) would be executed, but any and all commands after the first data entry prompt in that statement would be ignored without warning - there will be no display or indication of an illegal command.

- The **@S** command (for serial output) statement cannot be combined with any other command - even clear (**@Cx**) commands. After a **@S** command is successfully completed, the Base Station sends back to the host the ***RF Terminal ID*** followed by a ***CR*** (ASCII 13). There is a 231 character limit on data for this command. If you send a command of more than 231 characters, you will get an ***Illegal Command*** returned, (ID ? CR). If you need to send 300 characters of data, send the first part, wait for the acknowledgement (ID CR), and then send the remaining part.

  If you are using the **@S** command with a printer other than the Cameo or the QL3, you may have to set the **Protocol** parameter in the RF Terminal to XON/XOFF. This will allow the RF Terminal to deal with the character buffer limitations of your particular printer. If you are using the O'Neil MicroFlash Printer, you must send a NULL character before the valid data to wake up the printer. The QL/Cameo printers wake up when their DSR line is raised.

- The @M command is similar to the @S command, except it can be combined with other commands because it is a data entry command too. This command is for a printer initialization and magstripe input on the Zebra Cameo printer equipped with the magstripe option. The format of the command is:

  @Mdddatttta(EOT)

  where **dddatttta** might be **! U1 MCR 80 T1 T2+ CR + LF**

  (Refer to the Cameo manual for the exact string sequence you need to send. The above example sends over an 10 second request for reading Track 1 and Track 2). There is no reply to the host except the magstripe data. If the card cannot be read, pressing the ENTER key on the Terminal will send back ID+CR. This is the breakout method.

This command must be the last in a series of commands. For example, the following would be a typical multi-command statement:

@C0@1,1,0,Swipe Card@M! U1 MCR 80 T2 (CR)(LF)(EOT)

where **CR** is *ASCII 13*
LF is *ASCII 10*
EOT is *ASCII 4*

The statement causes the RF Terminal to transmit the string

!U1 MCR 80 T2 CR LF

to the Cameo printer. The printer then wakes up and blinks to indicate the magstripe input is ready to be swiped; when the swipe is complete, the Terminal sends back the data to the host computer as: **ID+T2:Data on Card+CR** (the printer's CR LF stuff is stripped). If the request is for Track 1 and Track 2, the data sent back is **ID+T1:data on 1+T2:data on 2+CR.**

- Every statement must end with a data entry "prompt" command, whether the statement is a single command by itself or several commands combined together. Any illegal statement will be ignored as a command but will be displayed on the addressed RF Terminal display exactly as written. If no Terminal ID was included in the statement, it will try to display the invalid statement on ID 0. Once the ENTER key is pressed on the Terminal displaying the invalid statement, the terminal sends the Base Station a "?" character. The Base Station in turn sends the message n?CR (where n is the Terminal ID and CR is a carriage return) back to the Host computer.

Here are some sample command statements:

@2,1,1,ENTER ITEM NO	Display ENTER ITEM NO on line 2, position 1 and wait for wait for data input
@V23@1,2,1,WRONG ITEM	Play voice message 23, display WRONG ITEM on line 1, position 2 and wait for data input
@C1@1,7,0,PICKING	Clear line 1. Display PICKING at position 7 of line 1. This command by itself is illegal. To be a valid statement, it must end with a data entry request. For example: **@C1@1,7,0,PICKING@2,7,1,ITEM**
@1,1,1,ITEM@2,1,1,QTY	Since only one command can be a "prompt" data entry request, this is an illegal statement and would be ignored as a command.

Base Station to Host Formats

The basic format of a message transmitted from Base to Host is simple:

Byte position	Function	Possible values
1	**RF Terminal ID**	***0-9, A-Z, a-z, - =***
2+	**Data Transmitted**	**
Last	**Termination of message**	***CR (ASCII 13)***

Typically, the Base Station is sending the "answer" to the hosts "question" - for example, if a Base sent a host message to a terminal #2 that said:

2@1,1,1,ITEM NUMBER + EOT

The RF Terminal would display ITEM NUMBER on line 1, position 1 and accordingly, the operator would then enter an item number by scanning or using the keypad. The RF Terminal transmits the data entered -say it's 123 - to the Base Station, which in turn transmits the following to the host:

2123+CR

Where **2**..... is the *Terminal ID*
123.....is the *data*
CR..... is the *termination*

Besides data, there are other messages that the Base Station will send to the Host:

Illegal Command

When a terminal receives an illegal statement from the host, it will display the entire statement on the terminal. Once the ENTER key is pressed on the terminal, the terminal sends a "?" back to the Base Station.

Byte position	Function	Possible values
1	**RF Terminal ID**	***0-9, A-Z, a-z, - =***
2+	**Illegal Command**	***?***
Last	**Termination of message**	***CR (ASCII 13)***

For example, if Terminal #2 received an illegal command, the Base station would transmit to the host:

2?CR

Serial Reply

After a Serial command (**@S**) has been successfully completed, the Base Station sends to the Host the *Terminal ID* followed by a *CR*. Serial commands are typically used for attached serial printers. Serial commands cannot be combined with other commands in a message to the Base Station /Terminal. Remember, you can only send 231 characters (including the *ID + @S + EOT*).

SIGN ON

To login to the host computer, the user presses a key on the RF Terminal at power-up to get to the **SIGN ON** screen. As the user **SIGN**s **ON**, the Base Station sends back the following **SIGN ON** message to the host:

Byte position	Function	Possible values
1	**RF Terminal ID**	***0-9, A-Z, a-z, - =***
2+	**SIGN ON**	***SYN*** (ASCII 22)if 6 line display configured as 6 line display. ***SI*** (ASCII 15) if 6 line display terminal configured as 4 line display.
Last	**Termination of message**	*CR* (ASCII 13)

After a terminal **SIGNS ON**, the host should be prepared to acknowledge the **SIGN ON** and give the terminal instructions, such as:

Standby for Assignment, Press ENTER to acknowledge
or
Nothing to do, Press ENTER and See Supervisor
or
Pick Item 1234

If there is something for the Terminal to do, the host should send instruction to the terminal (as in "Pick Item 1234" above). If there is nothing to do at the time of **SIGN ON**, the host should acknowledge the **SIGN ON** and tell the terminal to *Stand By* or *See Supervisor* (see lines 1 and 2 above). You will notice that in lines 1 & 2 above, there is a request for the operator to press the ENTER key. This is required for the message to be a valid command - remember that all messages must end with a request for data input. The host should then expect a response from the terminal of *Terminal ID + CR*.

SIGN ON is a good way for the terminal operator who has not received instruction from the host for several minutes to determine if he is still connected and if the host is still functioning. By **SIGN**ing **OUT** and **SIGN**ing back **ON**, the operator should receive a message that there is nothing to do. It is also a good idea for the host to keep track of elapsed time that a terminal has not had a message sent out to it. The host should then send a message periodically to reassure the operator (remember to ask him to press ENTER) that instruction is coming or tell him to see his supervisor for re-assignment (or whatever makes sense for your application).

Ideally, if the operator is leaving the area (to go to lunch or move to another building) before he is out of range of the network, he should **SIGN OUT**, then **SIGN ON** upon his return. An alternative is to have the operator press the END key followed by the ENTER key before he leaves. This could trigger a prompt from the host to send a message saying " Press BEGIN when ready again". Upon the operators

return, he would press the BEGIN key followed by the ENTER key, allowing him to receive a message immediately.

A 6 line display terminal configured as a 6 line display sends *ASCII 22* as its' **SIGN ON** character. A 6 line display terminal configured as a 4 line display will transmit the *ASCII 15* character for **SIGN ON**.

SIGN OUT

When a RF Terminal is powered down manually or the user presses the F1 key to exit data entry mode to go to one of the other modes (SETUP or ONE WAY), the host receives the following **SIGN OUT** message:

Byte position	Function	Possible values
1	**RF Terminal ID**	***0-9, A-Z, a-z, - =***
2+	**SIGN OUT**	***SO*** (ASCII 14)
Last	**Termination of message**	***CR*** (ASCII 13)

Addressing a Terminal not SIGNed ON

If the host attempts to send a message to a terminal that is not SIGNed ON, the Base Station sends back the following message to the host computer:

Byte position	Function	Possible values
1	**RF Terminal ID**	***0-9, A-Z, a-z, - =***
2+	**Terminal NOT Signed In**	***DC1***(ASCII 17)
Last	**Termination of message**	***CR*** (ASCII 13)

The ASCII 17 character can be changed to ASCII 16 for XON/XOFF sensitive systems by changing the Base Station Setup.

Sequence Error Message

The one-for-one "host prompt/terminal response" protocol must be observed by the host program at all times. The host cannot send a second data entry prompt until it has received a response to the first data entry prompt. If it does, this is considered a Sequence Error. If the Base Station receives a command that is out of sequence, it sends the following message back to the host:

Byte position	Function	Possible values
1	**RF Terminal ID**	***0-9, A-Z, a-z, - =***
2+	**Sequence Error**	***DC2*** (ASCII 18)
Last	**Termination of message**	***CR*** (ASCII 13)

If the Base Station receives 5 Sequence Errors in a row, it transmits the following message to the Terminal and shuts down:

Base Shut Down
Due to Host Logic
Error

Check your program for the sequence error before starting again. You will have to cycle power on the Base Station and have the Terminal **Sign On** again in order to continue.

Base Station Initialized Message

Whenever the Base Station is powered up, it sends the following message back to the host:

Byte position	Function	Possible values
1	**BASE ID**	* (Base ID is fixed)
2+	**Base Initialization**	***DC3*** (ASCII 19)
Last	**Termination of message**	***CR*** (ASCII 13)

Since ASCII 19 is XOFF, the ASCII 19 character can be changed to ASCII 20 for XON/XOFF sensitive systems by changing the ***Base Station Setup***. The **Base Station Initialized** message is provided so that the host will know that there has been a power interruption on the Base Station. When a serial device powers up, the first byte transmitted is often garbage. QBASIC handles the garbage character without incidence, but GWBASIC does not unless ON ERROR GOTO is used to trap the error. Be aware of this potential garbage-byte problem in your programming. To isolate and test for the problem, power up the Base without the serial cable connected. After you power the Base up, plug in the serial cable. You will not see the "**Base Initialized**" message but it should not matter when testing for the garbage data.

If a terminal is **SIGN**ed-**ON** to the system, and the base station is re-initialized, the following message is sent to the terminal:

Base Reinitialized X
Cycle Power on RF
Terminal and Sign-on
again to Restart_

where **X** is either a ***P*** (base initialization was power-related) or ***H*** (base initialization was host-related).

Transmitting ASCII characters using the terminal keypad

There are some keys on the RF Terminal keypad that when pressed, can transmit special ASCII characters back to the host program. This feature might be used by a programmer to allow the operator to review transactions. The keys are as follows:

Key	Code transmitted to Host
UP ARROW key	***FS*** (ASCII 28)
DOWN ARROW key	***GS*** (ASCII 29)
LEFT ARROW key	***RS*** (ASCII 30)
RIGHT ARROW key	***US*** (ASCII 31)
BEGIN key	***ETB*** (ASCII 23)
END key	***CAN*** (ASCII 24)
SEARCH key	***VT*** (ASCII 11)

The **STATUS** key is reserved to only display the ***Time*** and ***Date***.

These keys can be used without pressing the ENTER key by using the ***Control Keys Only*** Terminal Setup parameter.