Before you begin programming…

The "Application Host Program" or "Server Program" is a program designed and written by the user that incorporates the WD802Term ActiveX control. When an 802 RF Terminal is turned on and "Signs In" to the host, the WD802Term control reports a terminal ID number to the host application that is unique to that terminal. The IP address of a particular terminal can also be obtained.

Once an 802 RF Terminal is signed in, it receives messages from the host user program. The terminal responds back to the host application program with data that was keyed or scanned by the terminal's user. The host application program processes the data and sends back the next prompt. Each 802 RF Terminal has a unique IP address (or at least a unique Mac address). The host program can obtain any terminal's IP address or, for the sake of compatibility with programs written for 70/700 series terminals, resolve terminal addresses to a single character Terminal ID (0-9,A-Z, a-z, and -=) by using the WD802Term ActiveX component.

The dialog between 802 RF Terminal and Application Host is established when a terminal connects to the 802.11 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 an 802.11/b Access Point.

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 Server may return to you. The list includes:

  Sequence Errors detected
  Illegal Command detected
  Server Re-Initialized
  Addressing a Terminal Not Signed In
  Command without an ID

  All of these error conditions are detailed in the next chapter. Don't forget to program for them; this is a common mistake. Failure to trap them will give create very strange, unpredictable results.

  Even though you don't think your code will ever make a mistake, take advantage of feedback that the Server provides. Failure to do so is a common mistake that eventually results in serious program failure, sometimes due to hardware problems that go undetected.

- **Parse the Returned Strings thoroughly**. Don't assume anything about the next response from the Server to your program and look only for the partial string such as the ID only. Parse the string returned 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 Terminal) but try to create an application that will allow for easy expansion.

- **Use the Test Program**. The test program can at least allow you to see how the system functions and whether you can anticipate any system-wide problems. The test program should also be used as a response-time benchmark.

- **Study the Demo Programs**. Demo programs are included for examples of how to use the ActiveX tool provided.

Failure Planning

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 SIGNED OUT in mid-transaction, the Server 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.

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 *re-send* the last prompt to the Terminal when he SIGNs ON again.

Programming for the 802 RF Terminal

The programming support offered for the 802 RF Terminal is an ActiveX drop-in component. Every necessary function is defined - you just complete the code for each function.

The ActiveX component functions as a Server for managing data traffic to and from one or more terminals (your “terminal network”) and as an interface for your host application. There is no other software required, besides this control and your host application, to run a terminal network.

Introduction

It is important to note that creating working programs for the 802 Terminal requires significant programming skills. In order to create a working application you will not only need to be familiar with a programming language such as Visual Basic or Delphi or C++; you will need to be experienced and comfortable building real applications with one of these (or similar) programming environments.

You should be familiar with the concepts of objects, methods, event handlers, properties, scope, arrays, pointers, variable and value parameter passing, etc.

Database connectivity, data structures, and state management techniques are key to creating a program that can work with multiple terminals.

If you do not understand the references mentioned here and the implications of the pseudo code illustrations, you may not have the skills necessary to create a working application using the 802 Terminal WD802Term ActiveX programming tool.

What follows are just examples to get you started and show some basic technique. There are lots of different ways to do this and it is possible to create sophisticated transactions that intelligently instruct the terminal operator and collect complex data.

Objects, Properties, Methods, and Events

The programming model for the 802 Terminal is based on the WD802Term ActiveX component. This component is an "object" that is embedded in your application that you build using a compatible programming environment such as Visual Basic or Delphi.

The WD802Term "object" has a number of methods, event handlers, and properties that it publishes (makes available) to your program.

Properties are generally initialization and configuration settings that you set once when your application starts. Examples are ServerActive and LogFileName.

Methods are how you initiate communication with the WD802Term object. You can request that a prompt be sent to a terminal, map a terminal ID to an IP address, and a variety of other services.

Events are how the WD802Term object initiates communication with your application. You write "event handlers" in your program to respond that respond appropriately when WD802Term "fires" an event that activates its associated event handler. Examples are OnTermSignIn and OnTermData.

Creation of the source code "shell" for an event handler is generally handled by your development environment (IDE for Integrated Development Environment). For example, in Delphi, simply double click in an event field in the Object Inspector; or in VB, from the source code window, select the Object from the pull down list at the top left and then click on the desired event in the Procedures pull down list at the top right.

You will notice that the IDE creates a "skeleton" function (or procedure or sub) in source code that will include any parameters required to retrieve the data delivered to the event handler by the WD802Term object.

It is your job to add functionality to the skeleton event handler that is relevant to your application. It is important to remember to keep event handlers short and efficient. Do not make method calls to WD802Term from within a WD802Term event handler. Instead set up a state management and polling or threading scheme as outlined below.

Signing In, Data Structures, Transactions, and State Management

When a terminal **SIGNS ON**, it sends a signal to your application via the WD802Term object, which fires the OnTermSignIn event. When the event fires, the terminal has already been assigned an ID number by WD802Term and this ID number is passed to your application in the event handler.

Once it has signed in, the terminal is now waiting for your application to make a call to one of WD802Term's methods that sends a prompt to the terminal. Keep in mind that it is important to keep event handlers short and to try to avoid making method calls from inside an event handler. This brings us to transactions, state management, and data structures

Generally, you will have some kind of **transaction** process that you will define. Typically a transaction would be defined by a series of prompts and the data returned in response to the prompts.

Here is an example of a series of prompts that represent a transaction. When a transaction is completed, it repeats as controlled by the host application

Go to location XX
Press enter when ready:

Pick Item YY qty N
Scan barcode:

Enter Qty Picked:

Of course you can see how you might handle incorrect items or wrong quantities by having your application generate prompts such as

Incorrect Item
Scan correct item:

Too many items
Pick item YY qty N

Enter Qty picked:

These are just examples and there are other ways to design a transaction of this sort. The point is that you will need to create a **data structure** that defines the transaction and another data structure that keeps track of where in the transaction each terminal is (**state management**), independent of any other terminals that might be active.

You will need to create a data structure or record of some sort that keeps track of each terminal's state. The structure might keep track of where in a transaction process a terminal is; any pending data from, or prompt for, the terminal; and any other information that is relevant to a terminal in the transaction process you have defined.

An easy way to create a data structure for each terminal is to define an array of User Defined Type (VB) or Records (Delphi) or Structs (C++). The array would be large enough to accommodate all of your terminals (one array element for each terminal) and might look something like this in VB

```
Type TermData
        TermID as integer
        TransactionIndex as integer
        PendingPrompt as string
        PendingData as string
        ErrorCode as integer
        Transaction as TransactionData
        Etc.
End Type

dim Terminals(25) as TermData
```

You will also need to define a transaction. Typically the transaction is applied to all terminals so it is a single structure (there is not a copy for each terminal)

```
Type TransactionElement
    commandcode as integer
    promptline1 as string
```

```
    promptline2 as string
    promptline3 as string
    NextIfError as integer
    NextIfOK   as integer
End Type

dim TransactionSequence(10) as TransactionElement

TransactionSequence(1).commandcode = 1 'InputAny
TransactionSequence(1).promptline1 = "Go to Location " +
GetLocationCode
TransactionSequence(1).promptline2 = "Press Enter When
Ready "
TransactionSequence(1).NextIfError = 1
TransactionSequence(1).NextIfOK = 2
Etc.
```

As mentioned above, it is important to keep your event handlers short and efficient. This means that your program should spend as little time as possible "inside" the event handler (running the handler's code). Also, it is important to NOT call WD802Term methods from inside WD802Term event handlers. So a solution is needed that allows your program to respond to events outside of the event handlers.

An easy way to do this is with a polling scheme. Add a timer object to your application and in the timer's event handler check the **TermData** structure for the state of each terminal and make any appropriate WD802Term method calls from within the timer object's event handler. The timer can be set to fire every 100 milliseconds or every 10 seconds or anything in between that is appropriate for your application.

Another, more advanced way to decouple the WD802Term method calls from the event handlers is using threading techniques where each terminal is "running" in a thread that monitors that terminal's state.

WD802Term/ActiveX

WD802Term is a drop in ActiveX component that allows programmers to easily add the ability to send prompts to and receive data from their R/F Terminal across a wireless 802.11b Ethernet network connection.

The ActiveX component is compatible with Visual Basic, Visual C++, Delphi, and most other 32-bit development platforms.

Programming Considerations

Remember, plan for every error that the Server might return including:

Sequence Errors detected
Illegal Command detected
Addressing a Terminal Not Signed In

Programming languages that can interface with the Active X tool include VB, C++, Delphi, Access, FoxPro, etc.

If the Application Server receives any of:

1. 5 Addressing a Terminal not **SIGN**ed **ON** messages in a row or
2. 5 Sequence Errors in a row, or
3. 5 illegal commands in a row,

the Server transmits the following message to the Terminal and shuts down:

Server Shut Down
Due to Host Logic
Error

Check your program to correct these errors before starting again. The host application program will have to restart and you will have to cycle power on the Terminal and **SIGN ON** again in order to continue.

Network Setup

- The network settings on the server must support TCP/IP communications.
- It is critical that the Terminal(s) and Server are "visible" to each other across your network. Both must have an IP address in the same subnet. Both the Server and the Terminal Device(s) 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.
- **WD802Term** uses port 54140
- You can link server and terminal through a dial-up or DSL internet link as long as the server has a static IP address and your router passes the above port.
- If you are unsure of how to set up your IP configuration properly, refer to your network administrator for help.

Server Communications

- Run the 802 Test Program on the server computer. Now go to an 802 RF Terminal and attempt to SIGN IN to the 802 Test Program. If the terminal connects and responds with a prompt “Enter Data”, you are configured properly. Go to the server computer, shut down the 802 Test Program and begin work on your **WD802Term** server application.
- Before making any **WD802term** method calls in your application, make sure to turn the Server on by calling

  WD802Term.InitializeServer

  WD802Term.ServerActive = True

Test For Good Communication

- Implement an event handler for **OnTermSignIn** that causes a beep or displays a message when called. If communication between the host PC and the terminal is good, your event handler will fire when your program is running and you sign in a terminal on your network.

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 1000) is in its transaction sequence. Remember, **WD802Term** will give you a unique ID number for each Terminal that signs in and that number is "locked" to that Terminal until it signs out. 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 for more ideas.

- 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 **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 IP address (terminal ID conflict).

Control Keys for Possible Programming

There are some keys on the 802 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. **You can use these keys for special program functions, such as scrolling thru data, backing up steps, jumping, finishing a process, etc** 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.

The Control keys can be used *without* pressing the ENTER key by using the **Control Keys Only** Terminal Setup parameter. See **Chapter 2;** *RF System Setup* for details.

Concepts – WD802Term ActiveX

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 **ServerActive** 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 **www.barcodehq.com**

Properties – WD802Term ActiveX

Properties are the various configuration variables used by the **WD802Term** control. They are directly assignable in your application (eg. "WD802term.ServerOn = true") and can be set in your development environment's object browser.

Note that your development environment may show more properties for the **WD802Term** control than are listed here. This is normal. You may ignore properties you see listed in your development environment that are not listed here.

ActiveTerminal

Valid values: *0-999*

Function: This is the terminal ID (0-999) to which method call instructions are directed.

Terminal ID "number" is mapped to actual terminal IP address dynamically.

See method "GetIPAddress" to get terminal IP address associated with an ID.

Keep in mind that unless your terminal is configured with a static IP address, each time it is powered on it can have a different IP address (issued by your DHCP server). Also, regardless of the terminal's IP address, it can be assigned a different ID (by WD802Term) each time it Signs On to the server. Keep this in mind and design your application accordingly if you require specific physical terminals to perform specific individual tasks.

ServerPort

Valid values: *5000-65536*

Function: These are the IP ports used by WD802Term to pass command information to and from each terminal. This setting must match the PORT setting in all terminals used with this host application. Do not use port values less that 5000. If you are not familiar with IP ports, leave this value at the default setting.

ServerActive

Valid values: *True, False*

Function: Set ServerActive to True to begin listening for terminals. Before setting to True, be sure to call the InitializeServer method (See Methods)

LogFileName

Valid values: *Any valid file path and name*

Function: Leave blank if you do not want a log file kept. If you enter a filename here, WD802Term will create the file (or add to it if it already exists) in standard text file format that you can open using Windows Notepad.

LogFileSize

Valid values: *1000 through 2GB (2000000000)*

Function: This is the maximum log file size that WD802Term will keep. If the file exceeds this size, the oldest entries are removed to make room for new entries.

Quiet

Valid values: *True, False*

Function: If **Quiet** is set to *True* then any status and error message generated by **WD802term** will be suppressed.

Methods – WD802Term ActiveX

Methods are commands that you issue to the **WD802term** 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 **WD802term** control than are listed here. This is normal. You may ignore methods you see that are not listed here.*

InitializeServer

Parameters: *none*

Function: Prepares the Server to be started. Follow this call by setting the ServerActive property to True. This must be called before any of the methods described below.

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.

InputExtKeyBd

Parameters: *line, position, prompt*

Function: This instructs the ActiveTerminal to display the prompt at line and position and wait for data to be received from the PS/2 keyboard attached using an adaptor to the terminal serial port. Waiting for external keyboard 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). External keyboards are supported by terminals using firmware version RFU1010 or later.

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 "breakout" 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).

InputMagStripe

Parameters: *data*

Function: This command is for a printer initialization and magstripe input on the Zebra Cameo printer equipped with the magstripe option.

This instructs the ActiveTerminal to send the string data to an attached Cameo printer with the magstripe option 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).

Data 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 a 10 second request for reading Track 1 and Track 2). There is no reply to the host except the magstripe data.

OutputSerial

Parameters: *data*

Function: This instructs the ActiveTerminal to send data to the terminal's serial port. Data must be less than 232 characters in length for each call to OutputSerial. If you are sending data to a printer attached to the terminal, make sure to set the Protocol parameter in the 802 RF Terminal to XON/XOFF. See the 802 RF Terminal Manual for details.

Special Considerations:

- After an **OutputSerial** call is successfully completed, the terminal will return (as data) a CR (ASCII #13 Carriage Return). This will fire the **OnTermData** event. If there is a problem with the serial data you will see an error message at the client and in the log (if enabled). If the data string is too long, the **OnTermIllegalCommand** event will be fired.
- Do not call **OutputSerial** for the Terminal again until a return code is received.
- Do not call an Inputxxx method for the same Terminal until a return code is received.
- If you need to send more than 232 characters, send the first part, wait for the acknowledge (#13) and then send the next part.
- Calls to **OutputSerial** cannot be combined with other method calls.

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

Parameters: *none*

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

Parameters: *none*

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

ReInitAll

Parameters: *none*

Function: Instructs all attached terminals to re-initialize.

OutputRaw

Parameters: *data*

Function: This allows you to override all of WD802Term's Input methods (or any other method, for that matter) and send whatever data you want to the Active Terminal. This is most useful for adapting old DLL-based code to use the new ActiveX system.

MapTermID

Parameters: *TermNumber*

Function: Returns the actual terminal ID letter code for a given terminal number. Use the returned character to match with the Terminal ID programmed into a non-802 RF Terminal. This is provided for backwards compatibility and should not be used for new host applications.

GetIPAddress

Parameters: *TermNumber*

Function: Returns the actual IP address of the terminal associated with a given ID code.

GetErrCode

Parameters: *none*

Function: Returns code for the most recent error. Calling this method resets the Error Code to 0.

Error Codes

0. No Error
1. Command Data Too Long
2. Error on Close Device
3. Serial Out Data Too Long
4. Invalid Terminal ID On Last Command
5. Terminal ID Format Error

Events – WD802Term ActiveX

WD802term 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. For example, **OnTermData** passes the data that was keyed or scanned into the terminal.

Terminal ID is always passed as 0-999.

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

Before any **WD802term** events will fire, you must make sure to turn the Server on by calling

WD802Term.InitializeServer

WD802Term.ServerActive = True

OnServerActivate

Data passed: *none*

Event: Called when the ServerActive property is set to True immediately after the Server begins listening for terminals.

OnTermSignIn

Data passed: *terminal*

Event: A 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 WD802Term receives 5 sequence errors in a row, a Host Logic error is generated and the server shuts itself down.

While WD802Term/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.

WD802Term /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 WD802Term can respond to another keypress on the terminal. If you have already entered some data on the terminal and press an arrow key, this event will not fire.

OnTermDownArrow

Data passed: *terminal*

Event: The down-arrow button has been pressed on a terminal. You must issue another Input method call before WD802Term can respond to another keypress on the terminal. If you have already entered some data on the terminal and press an arrow key, this event will not fire.

OnTermLeftArrow

Data passed: *terminal*

Event: The left-arrow button has been pressed on a terminal. You must issue another Input method call before WD802Term can respond to another keypress on the terminal. If you have already entered some data on the terminal and press an arrow key, this event will not fire.

OnTermRightArrow

Data passed: *terminal*

Event: The right-arrow button has been pressed on a terminal. You must issue another Input method call before WD802Term can respond to another keypress on the terminal. If you have already entered some data on the terminal and press an arrow key, this event will not fire.

OnTermBeginKey

Data passed: *terminal*

Event: The BEGIN button has been pressed on a terminal. You must issue another Input method call before WD802Term can respond to another keypress on the terminal. If you have already entered some data on the terminal and press the Begin key, this event will not fire.

OnTermEndKey

Data passed: *terminal*

Event: The END button has been pressed on a terminal. You must issue another Input method call before WD802Term can respond to another keypress on the terminal. If you have already entered some data on the terminal and press the End key, this event will not fire.

OnTermSearchKey

Data passed: *terminal*

Event: The SEARCH button has been pressed on a terminal. You must issue another Input method call before WD802Term can respond to another keypress on the terminal. If you have already entered some data on the terminal and press the Search key, this event will not fire.

Portable Printers

Cameo and QL 320 Common Information

Both of these printers are stocked by Worth Data for the convenience of our users who need portable printing.

These printers do **not** require any special protocol; they do not require the "wake-up byte" as do other printers. They do require a special cable that can be ordered from Worth Data (part #C12); cable pin-outs are available in **Appendix C**: ***Cable Pin-outs***.

Shipped with every Cameo or QL 320 printer ordered is a CD ROM with the Programmers Manual in PDF format and a label design program – ***LabelVista***. This program allows you to design the program and create multiple format files that can be sent to the printer where they become resident in flash. Variable fields are defined and can then be filled in by the program when in operation.

Keep in mind the following information when using these versatile printers:

- The printer turn on ("Wake-up") is accomplished by the 802 RF Terminal toggling the DSR line on the printer, so only the @S command and the data you are sending to the printer is needed.
- Once the 802 RF Terminal has turned on the printer, it stays on until the host program turns it off using the POWER OFF COMMAND "ESC(0x1b)`p'(0x70)" described in the **Printing Systems Programming Manual**, or until the automatic shutdown takes place (2 minute default).
- The 231 character limit applies to your command string. See your **Portable Printing Systems Programming Manual** for details on programming your printer.

Zebra Cameo Printer

The Zebra "Cameo " Printers are portable direct thermal receipt printers, (**not** label printers – the QL 320 below prints labels). Bar codes can be printed on the receipts, but you can't print labels.

One model of the Cameo printer is available with a magnetic stripe reader, allowing magnetic stripe input to the 802 RF Terminal using the @M (magstripe input) command.

- The Cameo printer with magstripe input is capable of reading Track 1, Track 2 or Tracks 1&2. See your **Portable Printing Systems Programming Manual** for the correct character string to send in the @M command to turn on the magstripe reader. (see page 6-2 for details).

- When the Terminal sends data to the host, it sends it in the following format:

  Terminal ID + *DATA* **+ CR**

  Typically, the data is simply a string of characters, but in the instance of data coming from the magstripe reader, there are some additional characters you need to be aware of. The magstripe reader sends its data in the following formats:

  Track 1:

  T1: *DATA*

  Track 2:

  T2: DATA

  Track 1&2:

  T1: DATAT2: DATA

  So, when the 802 RF Terminal transmits the data to the host, it will be in the following format:

  Terminal ID + *T1: DATA* **+ CR**

  or

  Terminal ID + *T2: DATA* **+ CR**

  or

  Terminal ID +*T1: DATA* **+** *T2: DATA* **+ CR**

For further information, see your **Printing Systems Programming Manual** on the CD ROM shipped with the printer.

Zebra QL 320 Printer

The QL 320 Printer is used for label printing. It doesn't have Magstripe input. The classic application is for printing shelf labels during shelf price verification:

1. The operator scans a shelf label.
2. The Terminal transmits scanned data to the host computer.
3. The host computer looks up the price, description, etc. and transmits the computer price back and sends the necessary commands to the attached QL 320 printer to prints a new shelf label with the correct price.
4. The terminal operator then peels off the label and applies it to the shelf.

Each printer is shipped with a no charge roll of thermal paper that can be used for development, including determining the exact label size that best fits you needs and the capabilities of the printer.

We stock the 2” and 3” QL 320 printers with several label sizes immediately available including:

Part Number	Description	Price/ Roll
E2L1	2"x1" Vinyl Shelf Adhesive Labels	$3.50
E2L2	2"x1.25" Paper Permanent Adhesive Labels	$2.50
E2L3	2”x2” Paper Permanent Adhesive Labels	$2.50
E2L4	2”x1.25” Vinyl Shelf Adhesive Labels	$3.00
E3L1	3"x1" Vinyl Shelf Adhesive Labels	$7.50
E3L2	3"x1.75" Paper Permanent Adhesive Labels	$5.00

Shelf adhesive labels are designed for ease of removal to facilitate replacement. Permanent adhesive labels are designed to stick and stay stuck, making removal difficult without leaving a residue.