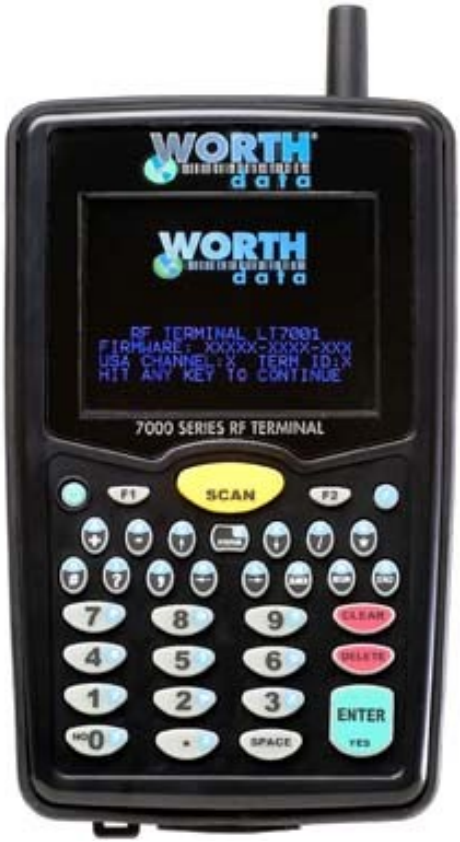
ActiveX Programming

For The

7000 RF Terminal



WORTH data

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: The *WDterm* ActiveX 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.

Color Codes for 7001 (15-line) Terminal

0 - aqua
1 - black
2 - blue
3 - fuchsia
4 - gray
5 - green
6 - lime
7 - maroon
8 - navy
9 - olive
10 - purple
11 - red
12 - silver
13 - teal
14 - white
15 - yellow

OpenDevice *(All Terminal Versions)*

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 *(All Terminal Versions)*

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.

DefineFormat *(15-Line Terminal Only)*

Parameters: font, linecount

Function: This adds a line formatting definition. This command is called multiple times to build a display formatting definition for multiple lines which is then sent to the *ActiveTerminal* by the *SendFormat* command.

Font is an integer code: 0="small", 1="medium", 2="large"

Linecount is a number 1-15 indicating the number of lines to apply the font selection to.

If only one line is defined (that is *DefineFormat* is called only once with a linecount of "1"), then after *SendFormat* is called, only one line will be available for display on the *ActiveTerminal*.

There are a limited number of lines available depending on the font size(s) chosen. Each font has a defined height:

- small: 16
- medium: 24
- large: 32

The total height of the defined lines cannot exceed 240. If it does, an error code is generated (see *CheckError*) and the *SendFormat* command is ignored.

There is limited display width available for text. Depending on the font you select for a line:

- small: 26 characters
- medium: 20 characters
- large: 13 characters

If you try to send prompt or display text longer than this, it will be truncated and an error code is generated (see *CheckError*).

Must be followed by a "*SendFormat*" method call and then an "*Inputxxx"* method call to take effect.

SendFormat *(15-Line Terminal Only)*

Parameters: FG, BG

Function: Sends to the *ActiveTerminal* the current Format Definition as created by one or more calls to the *DefineFormat* method. It also sets the user-default display foreground and background colors. *FG* and *BG* are *ForeGround* and *BackGround* colors for the 7001 (15-line) terminal (See ***Color Codes***).

Must be followed by an "*Inputxxx"* method call to take effect.

InputAny *(All Terminal Versions)*

Parameters: line, position, prompt, shifted, timestamped

InputAnyColor *(15-Line Terminal Only)*

Parameters: line, position, prompt, shifted, timestamped, FG, BG

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. *FG* and *BG* are *ForeGround* and *BackGround* colors for the 7001 (15-line) terminal (See ***Color Codes***).

InputKeyBd *(All Terminal Versions)*

Parameters: line, position, prompt, shifted, timestamped

InputKeyBdColor *(15-Line Terminal Only)*

Parameters: line, position, prompt, shifted, timestamped, FG, BG

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. *FG* and *BG* are *ForeGround* and *BackGround* colors for the 7001 (15-line) terminal (See ***Color Codes***).

InputExtKeyBd *(All Terminal Versions)*

Parameters: line, position, prompt

InputExtKeyBdColor *(15-Line Terminal Only)*

Parameters: line, position, prompt, FG, BG

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. *FG* and *BG* are *ForeGround* and *BackGround* colors for the 7001 (15-line) terminal (See *Color Codes*).

InputScanner *(All Terminal Versions)*

Parameters: line, position, prompt, allowbreakout, timestamped

InputScannerColor *(15-Line Terminal Only)*

Parameters: line, position, prompt, allowbreakout, timestamped, FG, BG

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. *FG* and *BG* are *ForeGround* and *BackGround* colors for the 7001 (15-line) terminal (See ***Color Codes***).

InputYesNo *(All Terminal Versions)*

Parameters: line, position, prompt

InputYesNoColor *(15-Line Terminal Only)*

Parameters: line, position, prompt, FG, BG

Function: This instructs the **ActiveTerminal** to display the *prompt* at *lin*e 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.

FG and *BG* are *ForeGround* and *BackGround* colors for the 7001 (15-line) terminal (See ***Color Codes***).

InputPassword *(All Terminal Versions)*

Parameters: line, position, prompt, shifted

InputPasswordColor *(15-Line Terminal Only)*

Parameters: line, position, prompt, shifted, FG, BG

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. *FG* and *BG* are *ForeGround* and *BackGround* colors for the 7001 (15-line) terminal (See ***Color Codes***).

InputSerial *(All Terminal Versions)*

Parameters: line, position, prompt

InputSerialColor *(15-Line Terminal Only)*

Parameters: line, position, prompt, FG, BG

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). *FG* and

BG are *ForeGround* and *BackGround* colors for the 7001 (15-line) terminal (See ***Color Codes***).

OutputSerial *(All Terminal Versions)*

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 RF Terminal to XON/XOFF. See the RF Terminal Manual for details.

Special Considerations:

- After an *OutputSerial* call is successfully completed, the Base Station will return (as data) a CR (ASCII #13 Carriage Return) for the terminal. 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 same Base Station and Terminal again until a return code is received.
- Do not call an *Inputxxx* method for the same Base Station and 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
- See the RF Terminal Manual for details.

SendDisplay *(All Terminal Versions)*

Parameters: line, position, prompt

SendDisplayColor *(15-Line Terminal Only)*

Parameters: line, position, prompt, FG, BG

Function: This instructs the **ActiveTerminal** to display the *prompt* at *line* and *position*. Must be followed by an "**Input**" ***method*** call to take effect. *FG* and *BG* are *ForeGround* and *BackGround* colors for the 7001 (15-line) terminal (See ***Color Codes***).

ClearScreen *(All Terminal Versions)*

Function: This instructs the **ActiveTerminal** to clear its display. Must be followed by an "**Input**" method call to take effect. Does not work on older 4-line terminals. Instead, use multiple *ClearLine* calls.

ClearLine *(All Terminal Versions)*

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 *(All Terminal Versions)*

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 *(All Terminal Versions)*

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 *(All Terminal Versions)*

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 *(All Terminal Versions)*

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

NOTE: *Base Stations using EEPROM versions prior to 9079 will cause the message "Base Reinitialized..." to be displayed on the terminal. Only the terminal has actually been re-initialized. Later Base Stations use the message "Buffer Reinitialized..." to indicate a single terminal re-initialization.*

ReInitAll *(All Terminal Versions)*

Function: Instructs all attached terminals to re-initialize.

OutputRaw *(All Terminal Versions)*

Parameters: data

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

MapTermID *(All Terminal Versions)*

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 RF Terminal.

GetErrCode *(All Terminal Versions)*

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
6. Display Formatting Error (15-line terminal only)

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.

NOTE: If you are experiencing problems with exception errors, make sure you are not calling any WDterm methods from inside WDterm event handlers. Instead use the event handlers to set program variables that are monitored elsewhere (perhaps in a timer event handler). Call WDterm methods from the monitoring function.

OnTermBaseRegister

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

OnTermSignIn15

Data passed: terminal
*Event:*A 15-line terminal has signed in. Terminal ID is passed in *terminal.*

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. If you have already entered some data on the terminal and press an arrow key, this event will not fire. See the programming section in the RF Terminal manual for details.

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. If you have already entered some data on the terminal and press an arrow key, this event will not fire. See the programming section in the RF Terminal manual for details.

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. If you have already entered some data on the terminal and press an arrow key, this event will not fire. See the programming section in the RF Terminal manual for details.

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. If you have already entered some data on the terminal and press an arrow key, this event will not fire. See the programming section in the RF Terminal manual for details.

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. If you have already entered some data on the terminal and press the Begin key, this event will not fire. See the programming section in the RF Terminal manual for details.

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. If you have already entered some data on the terminal and press the End key, this event will not fire. See the programming section in the RF Terminal manual for details.

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. If you have already entered some data on the terminal and press the Search key, this event will not fire. See the programming section in the RF Terminal manual for details.