COM Development Library

Version 3.7 Users Guide

Revised 5/21/03

Wavelink Studio Users Manual

Revision 5/21/03

Table of Contents

About This Document

We are very interested in improving the WaveLink Development Library documentation and welcome all criticisms and suggestions for improvement. Please direct them to:

WaveLink Development Library Documentation

11335 NE 122nd Way / #115
Kirkland, WA 98034
Phone:(425) 823-0111
Fax:(425) 823-0143
Email:documentation@wavelink.com

Reference Typographical Conventions

Italic type is used to represent placement of variable information. For example, the syntax for the GetEvent method is: *pszEvent* = *object*.GetEvent (). For this particular function "pszEvent" represents the variable for the input value returned by GetEvent while "object" represents the name of the method's parent object.

All syntax variable names, whether a return value or function parameter, will be prefixed with lower case Hungarian programming notation. The lower case character notation will be used to represent the specific data type for the variable. For example, the "Event" variable of the GetEvent method is prefixed with the lowercase characters "psz". This indicates that the "Event" variable should be passed as "string" type data within your specific development environment.

The notation used within this manual and the equivalent data types are as follows:

*b*Bool

*d*DWORD

*n*Integer

*psz*String

*l*Long

NOTE These conventions are not used in the WaveLinkWidget Extension.

About the WaveLink Development Library

The WaveLink Development Library is a Dynamic Link Library that you include into your OLE compliant development environment. Once included, you will have full access to eight objects and hundreds of methods designed specifically for wireless application development. These methods and objects range from basic input and output functions over a wireless network to playing custom tone files on an RF device.

Including the WaveLink Development Library

Before you begin developing custom wireless applications, the WaveLink Development Library must first be included into your specific development environment. This will provide access to the custom objects and methods necessary for wireless application development.

The WaveLink Development Library is easy to include and is done in the same way one includes any other OLE automated server. Simply reference the necessary WaveLink Development Library files with your development environment. These files are located in the Windows System directory. For Windows 95/98, this directory will be the Windows\System directory. For Windows NT, this will be the \System32 sub directory of the directory in which Windows NT was installed. Please reference the documentation included with your development environment for more specific information on including OLE automated servers.x

If you are using a C++ development environment, two additional steps are necessary to use the WaveLink Development Library.

First, you must make sure that all of the WaveLink header files are referenced by your development environment. The header files are stored within the \Development\Include directory of the WaveLink Studio COM install path. Please see the documentation included with your development environment for more information on including header files.

Second, to use the WaveLink Application Wizard you must copy certain .AWX files from the WaveLink Studio COM templates directory to your Visual C++ templates directory. If you are using Visual C++ 5.0, copy the file "WW50.AWX" from the "Templates" directory found within your WaveLink Studio COM install path to the "\Shared\DE" directory within your Visual C++ install path. If you are using Visual C++ 4.X, copy the two files

"WaveLinkWiz.AWX" and "OLEAppWiz.AWX" to your "Templates" directory within your Visual C++ install path.

Including WaveLink Objects Within Your Projects

Adding WaveLink objects to any of your project is a relatively simple task.

In a C++ environment, you may use the WaveLink OLE Wizard to create a basic wireless application template. Once created, add a new OLE based type library class by selecting the *WaveLinkOLE.tlb* file from the *\Development\Lib* directory of the WaveLink Studio COM install path. From this library you may then access any of the WaveLink Development Library objects. Please see the documentation included with your C++ development environment for more specific information on creating new OLE type library classes within your applications.

Note: The WaveLink OLE Wizard automatically creates an instance of all WaveLink objects for any new project. Do NOT add these objects again.

In other development environments, such as Visual Basic, you may simply reference the WaveLink type library.

Before developing wireless applications, you must include the Wavelink COM Development Library in your specific development environment. This provides access to the custom objects and methods necessary for wireless application development.

Including WaveLink Objects in Visual Basic 6.0 Projects

To access the Wavelink Development Library in Visual Basic, you reference it in your project.

To include the Wavelink Development Library in a Visual Basic Project:

1. Open the Visual Basic project to which you want to add the Wavelink Development Library.

2. Remove the default form.

   In the Visual Basic Project window, right-click the default form and select `Remove Form1`.

3. Add a module to the project.

In the Visual Basic Project window, right-click the project and select `Add > Module`.

4 Reference the Wavelink Development Libraries

From the **Project** menu, select `References`.

Enable the **WaveLink** and **WavelinkOLE** checkbox and click `OK`.

5 In your project, declare any objects you plan to use.

Example: `Public wlio As New RFIO` creates a public instance of the RFIO Object.

Including Wavelink Objects in other COM Languages

You can access the Wavelink Development Library in any COM-based language such as C++. The libraries are located in the following directory:

`<installpath>\Include\Libs`

In your programming environment, reference the Wavelink and WavelinkOLE type libraries and dynamic link libraries (.dll file extension) as instructed in the documentation for your specific programming environment.

Programming Considerations

For the most part, developing WaveLink wireless applications is no different than developing any other conventional application. Certain programming considerations must be kept in mind, though, to gain maximum performance from any WaveLink applications.

RF Display Size

The first programming consideration that must be taken into account is the limited and various display sizes of RF devices.. Any information, either input or output, placed outside of these display limitations will not appear. Therefore, you must always keep in mind the display limitations of the RF devices that will be using your application when designing application screens. You may use the RFTerminal object to obtain display characteristics of the RF device. If the RF devices on your network use different screen sizes, you may want to use the RFTerminal object to dynamically position screen output. For example, you can program screen output one row from the bottom of the screen in lieu of setting a static row number.

Error Handling

The second programming consideration of WaveLink application development concerns function error checking.

You can check for returned error values when calling a WaveLink Development Library method within your application. It is of *absolute* importance to check for returned error values after making an input call such as RFInput or GetEvent. The reason for this is that proper functioning of your wireless applications is dependent upon certain returned error values.

For example, on a disconnect command from the WaveLink Administrator a specific error message is returned to your application. This error message is returned to the first WaveLink function that communicates with the mobile device (for example, an RFIO method) before the device is physically disconnected from the network. Every subsequent WaveLink function following disconnect will then return the error message. Upon receiving this error message, you must instruct the application to exit. If this error message is not trapped at this time, the application can get caught in a loop and consume remaining system resources while waiting for further user input from a disconnected device.

File Extensions

The third programming consideration concerns the use of file extensions used with various WaveLink objects. Certain WaveLink objects will save files to RF devices using the following extensions. The extensions and their associated objects are as follows:

RFIO Object.	.SCR
RFMenu Object.	.MNU
RFTone Object.	.TON
RFBarcode Object.	.BAR
RFAuxPort Object.	.CFG

When using these specified objects, it is not necessary to include these extensions when referencing files of the object types. If you reference these files using the RFFile object, though, it will be necessary to include the appropriate file extension (or ".*" for all file extensions).

User Environment Variables

The fourth and final programming consideration that must be taken into account when developing WaveLink applications concerns user environment variables. The WaveLink Server creates a set of environment variables for each connecting RF user. These WaveLink Server environment variables contain the same information about the RF user's device as that within the RFTerminal object. Since much of the information within these environment variables will remain static during a user session, it is possible to use certain device values without making a RF call as would be the case if you were to request the same information using one of the RFTerminal methods.

For example, the WaveLink Server creates a user environment variable that contains the device display width. Since a device's maximum display width will not change you can limit network traffic by using the width value from the WaveLink Server environment variable rather than making an RF call during the creation of an RFTerminal Object.

Note: It is recommended that you use the RFTerminal methods to return the most current values for variables that may not remain static, such as a device's key state. The names of those particular variables are proceeded by an asterisks.

The Wavelink Server environment variables and their possible return values are as follows:

Table 0-1: *Wavelink Server Environment Variables*

Variable Name	Type	Description/Return Values
TERMTYPE	int	Raw terminal type. 1 – LRT 2 – VRC/PRC 3 – PDT 4 – WSXX 5 – PDT 68XX 6 – PDT61XX 11 – Palm Pilot 27 – CE, 2740 72 – CE, 7200 75 – CE, 7540 93 – Percon Falcon 5020 – Intermec Device 8000 – Windows Device 8001 – PPC 8002 – HPC 8003 – HPC Pro
TERMWIDTH	int	Terminal screen width in characters

Table 0-1: *Wavelink Server Environment Variables*

Variable Name	Type	Description/Return Values
TERMHEIGHT	int	Terminal screen height in characters
*TERMMAIN	int	Main battery state 0 - Good battery 1 - Dead battery
*TERMLITHIUM	int	Lithium battery state 0-Dead battery 1 - Good battery 2-Unit has no lithium battery
*TERMKEYSTATE	int	Terminal keyboard state 0 - Normal keyboard mode 64 - Capslock enabled
*TERMTIMEOUT	int	Keyboard timeout in seconds
*TERMBACKLIGHT	int	Backlight timeout in seconds
*TERMCURSORMODE	int	Hardware or software cursor status 0 - Hardware cursor 1 - Software cursor
*TERMCURSORSTART	int	Cursor start line
*TERMCURSOREND	int	Cursor end line
TERMID	string	Terminal ID Spectrum One - Terminal Id number Spectrum 24 - Terminal's IP address
TERMVERSION	string	WaveLink Studio Client version installed on RF device
TERMDISK	long	Total device disk space capacity
TERMMEMORY	long	Total device memory capacity

Additionally, the WaveLink Program Manager will also create two environment variables for each spawned application.

IMPORTANT NOTE: These environment variables are only available if your users logged in using the WaveLink Program Manager. These variables will not be available for your own custom applications.

The values for the WaveLink Program Manager environment variables are as follows:

Table 0-1:

Variable Name	Type	Description/Return Values
USERNAME	string	The current user's name
USERSIGNON	string	The current user's sign on

Note: You may add or remove additional environment variables to the WaveLink Program Manager at will. A recompile of the application WaveLinkPM.exe is all that is required.

Objects

The WaveLink Development Library includes the following objects:

RFAuxPort Object

RFBarcode Object

RFError Object

RFFile Object

RFIO Object

RFMenu Object

RFTerminal Object

RFTone Object

WaveLinkFactory Object

WaveLinkMenubarInfo Object

WaveLinkScribblePad Object

WaveLinkWidget Object

WaveLinkWidgetCollection Object

RFAuxPort Object

The RFAuxPort object is supported on ALL devices.

The RFAuxPort object provides bi-directional access to the serial port of an RF device.

Methods

ConfigurePort
QueryPort
RFGetLastError

Prog IDs

"WAVELINKOLE.RFAUXPORT"

Remarks

You must first configure a serial port using the ConfigurePort method before using the QueryPort method.

Note: The main communication port between the server and the client is configured and accessed automatically. The RFAuxPort object is provided for the auxiliary port on the RF device, typically a printer port.

Method Summary

ConfigurePort Method
- Configures UART settings for device's serial port.

QueryPort Method
- Sends and receives data to and from device's serial port.

RFGetLastError Method
- Returns value of last generated RFAuxPort error.

ConfigurePort Method

This member of RFAuxPort is supported on ALL devices.

The ConfigurePort method allows you to configure the basic UART settings for a RF device's serial port before calling the QueryPort method.

VB

```
bStatus  = object.ConfigurePort (nPort, nBaud, nDataBits,
nStopBits, nParity, nFlowControl)
```

VC++

```
HRESULT hr = object->ConfigurePort(short nPort, short nBaud,
short nDataBits, short nStopBits, short nParity, short
nFlowControl, BOOL *bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return any error value.

Parameters

nPort	The serial port of the RF device which UART settings you wish to configure (WLCOM1 or WLCOM2).
nBaud	The UART Baud rate of the serial port (see Remarks).
nDataBits	The UART data bit setting of the serial port (see Remarks).
nStopBits	The UART stop bit setting of the serial port (STOPBITS1 or STOPBITS2).
nParity	The UART parity setting of the serial port (see Remarks).
nFlowControl	The UART flow control setting of the serial port (see Remarks).

Remarks

A serial port must first be configured using the ConfigurePort method before the port may be queried for input using the QueryPort Method.

The possible values for nPort are:

WLCOM1	- Com port 1
WLCOM2	- Com port 2

The possible values for nBaud are:

BAUD150	- 150 BBS Baud
BAUD300	- 300 BBS Baud
BAUD600	- 600 BBS Baud
BAUD1200	- 1200 BBS Baud
BAUD1350	- 1350 BBS Baud
BAUD2400	- 2400 BBS Baud
BAUD4800	- 4800 BBS Baud
BAUD9600	- 9600 BBS Baud
BAUD19200	- 19200 BBS Baud
BAUD38400	- 38400 BBS Baud
BAUD110000	- 110000 BBS Baud

The possible values for nDataBits are:

DATABITS5	- 5 data bits
DATABITS6	- 6 data bits
DATABITS7	- 7 data bits
DATABITS8	- 8 data bits
DATABITS4	- 4 data bits

The possible values for nStopBits are:

STOPBITS1	- 1 stop bit
STOPBITS2	- 2 stop bits

The possible values for nParity are:

PARITYEVEN	- Even parity
PARITYODD	- Odd parity
PARITYMARK	- Mark parity
PARITYSPACE	- Space parity
PARITYNONE	- No parity

The possible values for nFlowControl are:

NOFLOWCTL	- No flow control
SOFTWAREFLOWCTL	- Software based flow control
HARDWAREFLOWCTL	- Hardware based flow control

See the Constant Values for the numeric equates returned by the function.

Example

See RFAuxPort Samples.

QueryPort Method

This member of RFAuxPort is supported on ALL devices.

The QueryPort Method lets you query a RF device's serial port for input. You may also use the QueryPort Method to send data to the RF device's serial port.

VB

```
pszResponse = object.QueryPort (nPort, nStart, nEnd,
nMaxLength, nTimeout, pszRequest)
```

VC++

```
HRESULT hr = object->QueryPort(short nPort, short nStart,
short nEnd, short nMaxLength, short nTimeout, LPCTSTR
pszRequest, LPCTSTR *pszResponse);
```

Return Value

pszResponse	Any returned response received from the RF device's serial port.

Parameters

nPort	The communicating serial port of the RF device (WLCOM1 or WLCOM2).
nStart	The starting character for returned data that is framed. Any data received prior to the starting character will be discarded. Enter the decimal equivalent of the ASCII character you wish to designate as the starting character of a data frame. If "WLNOCHAR is entered, no starting character will be defined.
nEnd	The ending character for returned data that is framed. Input will terminate immediately after the ending character is received. Enter the decimal equivalent of the ASCII character you wish to designate as the ending character of a data frame. If WLNOCHAR is entered, no ending character will be defined. A ending character may be defined without a starting character.
nMaxLength	The maximum number of characters that may be received before input is automatically returned from the serial port query. If WLNOMAXLENGTH is entered, no maximum length will be enforced.

nTimeout	The maximum amount of time (in seconds) that may pass before input is automatically returned from the serial port query. If WLWAITFOREVER is entered, no timeout value will be enforced. If zero (0) is entered any data that is immediately available will be read and returned.
pszRequest	Data which will be sent to the serial port before any query input is returned. Set this value to null if you do not wish to send data to the serial port before it is queried. The pszRequest field is often useful for sending data to devices which require a triggering event before responding, such as a scale.

Remarks

A serial port must first be configured using the ConfigurePort Method before the port may be queried for input using the QueryPort Method. This method may also be used with RF based printers when greater control over communications is required.

The possible values for nPort are:

WLCOM1	- Com port 1
WLCOM2	- Com port 2

Example

See RFAuxPort Samples.

RFGetLastError Method

This member of RFAuxPort is supported on ALL devices.

The RFGetLastError method returns the value of the last generated error.

VB

```
dError = object.RFGetLastError()
```

VC++

```
HRESULT hr = object->RFGetLastError(DWORD *dError);
```

Return Value

dError	The numeric DWORD value of the last generated function error.

Remarks

Whenever a function error occurs, an error value is generated. You may use RFGetLastError to return the value of the error. The generated errors are as follows:

WLNOERROR	No error returned from function call.
WLCOMMERROR	RF network communication error
WLMEMORYERROR	Memory allocation error
WLNOTINITIALIZED	RF object is improperly initialized.
WLREQUESTFAIL	Input request string format failure
WLINVALIDPARAMS	Invalid parameter(s) for function call
WLINVALIDRETURN	Invalid return designator for function call
WLFUNCTIONFAILED	RF device returned a failure code for the function
WLBCPARSEERROR	Failure parsing barcode file
WLBCADDERROR	Failed to add barcode to object
WLBCCLEARERROR	Failed to clear barcode list
WLBARCODELIMITEXCEEDED	Too many barcodes for configuration file
WLTONEADDERROR	Failed to add tone to object
WLTONECLEARERROR	Failed to clear tone list

WLIOQUEUEERROR	IO queue is empty
WLNOINPUTTYPE	Unable to determine input type
WLPORTTIMEOUT	Auxiliary port timeout
WLDTOUTOFSYNC	Terminal date/time out of sync by more than five seconds

See the Constant Values for the numeric equates returned by the function.

Example

See RFAuxPort Samples.

RFAuxPort Samples

Using Visual Basic sample code, the following application demonstrates the use of the RFAuxPort object and its methods. Both applications will configure a RF device's serial port with generic UART settings then send the string AS1T to the device connected to the serial port. The application will then wait ten seconds for a carriage return, then return any response.

```
' VB Sample Code
' RFAuxPort Object Demo Application
'
  dim wlaux as new RFAUXPORT
  dim sAns as String
  dim nPort as Integer
  nPort = 0 ' COM1
  Dim nError As Integer
  ' You MUST set up the port before first use
  if wlaux.ConfigurePort
          (nPort,BAUD19200,DATABITS7,STOPBITS1, _
          PARITYEVEN,HARDWAREFLOWCTL) = False Then
          ' Error Can't set port
  end if
.
.
.
  sAns=wlaux.QueryPort(nPort,0,13,-1,10,"AS1T")

  nError = wlaux.RFGetLastError
  If nError <> WLNOERROR Then
    GoTo ExitApp
  Else
  ' process sAns
```

RFBarcode Object

The RFBarcode object is supported on ALL devices.

The RFBarcode object lets you define a list of barcode configurations for use within your applications. Each individual barcode configuration defines a valid or invalid barcode for input based upon type, state, and length. RFBarcode objects may also be saved as barcode files directly to an RF device's memory for future use.

Methods

AddBarcode
BarcodeCount
BarcodeFileCount
BarcodeFileName
ClearBarcodes
Default
DeleteBarcodeFile
GetBarcodeFile
GetBarcodeType
GetDecode
GetExpand
GetMaxLength
GetMinLength
ListBarcodeFiles
PullBarcode
PushBarcode
RemoveBarcode
RFGetLastError
SetBarcodeType
SetDecode
SetExpand
SetMaxLength
SetMinLength
StoreBarcode

Prog IDs

"WAVELINKOLE.RFBARCODE"

Remarks

Individual barcode configurations are stored within an RFBarcode object. Each barcode is defined by type, decode state, expand state, minimum length, and maximum length. Once defined, a specific barcode configuration will remain within an RFBarcode object until either the object is released, the barcode configuration is removed from the object using the RemoveBarcode method, or all barcode configurations are cleared from the object using the ClearBarcodes method. This allows you to use a single barcode object within your application that may be modified rather than creating a new barcode object for each individual instance.

Method Summary

AddBarcode Method
- Adds a barcode configuration.

BarcodeCount Method
- Returns the total number of barcode configurations.

BarcodeFileCount Method
- Returns the total number of barcode configuration files.

BarcodeFileName Method
- Returns the name of a barcode configuration file.

ClearBarcodes Method
- Clears all barcode configurations.

Default Method
- Returns the default decode state for barcodes not explicitly defined within the current RFBarcode object.

DeleteBarcodeFile Method
- Removes barcode files from the RF device.

GetBarcodeFile Method
- Retrieves a barcode configuration file into the current RFBarcode object.

GetBarcodeType Method
- Returns the barcode type of a barcode configuration.

GetDecode Method
- Returns the decode state of a barcode configuration.

GetExpand Method
- Returns the expand state of a barcode configuration.

GetMaxLength Method
- Returns the maximum input length of a barcode configuration.

GetMinLength Method
- Returns the minimum input length of a barcode configuration.

ListBarcodeFiles Method
- Returns the total number of barcode configuration files and stores the list in the current RFBarcode object.

PullBarcode Method
- Restores a barcode file as the RF device's default barcode configuration.

PushBarcode Method
- Stores the RF device's default barcode configuration to a file for later restoration.

RemoveBarcode Method
- Removes a barcode configuration

RFGetLastError Method
- Returns value of last generated RFBarcode error.

SetBarcodeType Method
- Specifies the barcode type of a barcode configuration.

SetDecode Method
- Specifies the decode state of a barcode configuration.

SetExpand Method
- Specifies the expand state of a barcode configuration.

SetMaxLength Method
- Specifies the maximum input length of an existing barcode configuration.

SetMinLength Method
- Specifies the minimum input length of an existing barcode configuration.

StoreBarcode Method
- Stores the current RFBarcode object as a file on the RF device.

AddBarcode Method

This member of RFBarcode supported on ALL devices.

The AddBarcode method adds an additional barcode configuration to the RFBarcode object.

VB

```
bStatus = object.AddBarcode (nType, bExpand, nDecode, nMin,
nMax)
```

VC++

```
HRESULT hr = object->AddBarcode(short nType, BOOL bExpand,
short nDecode, short nMin, short nMax, BOOL *bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return any error value.

Parameters

nType	The barcode configuration type (see Remarks).
bExpand	The Expand state for the barcode configuration (True or False). This parameter is only valid with the UPC_E0 barcode type which may or may not be further expanded following decode. For all other barcode types pass this parameter as False.
nDecode	The decode state for the barcode configuration (DECODEON or DECODEOFF).
nMin	The minimum acceptable length in characters for the barcode configuration.
nMax	The maximum acceptable length in characters for the barcode configuration.

Remarks

The individual barcode configurations are stored within an RFBarcode object using a zero based index, that is the fourth barcode configuration has the index value of 3. Once defined, a specific barcode configuration will remain within an RFBarcode object until either the object is released, the barcode configuration is removed from the array using the RemoveBarcode Method,

or all barcode configurations are cleared from the object using the ClearBarcodes Method. This allows you to use a single barcode object within your application that may be modified rather than creating a new barcode object for each individual instance.

To add an additional barcode configuration to a saved barcode file, first make the barcode file the current RFBarcode object using the GetBarcodeFile Method, then use AddBarcode.

Note: Input length restrictions for barcode configurations can be eliminated by setting both the *nMax* and *nMin* parameters to zero(0).

The possible values for nType are:

CODE_39

UPC_A

UPC_E0

EAN_13

EAN_8

CODE_D25

CODE_I25

CODABAR

CODE_128

CODE_93

CODE_11

MSI

UPC_E1

WLNOSYMBOLOGY

The possible values for nDecode are:

DECODEON	The barcode configuration will be decoded when scanned and is valid for input.
DECODEOFF	The barcode configuration will not be decoded when scanned and will not be accepted for input.

Example

See the StoreBarcode Method.

BarcodeCount Method

This member of RFBarcode is supported on ALL devices.

The BarcodeCount method returns the number of individual barcode configurations within an RFBarcode object.

VB

```
nBarcodes = object.BarcodeCount ()
```

VC++

```
HRESULT hr = object->BarcodeCount(short *nBarcodes);
```

Return Value

nBarcodes	The barcode configuration count variable. This method will return a -1 if an error occurs. Use the RFGetLastError method to return the generated error code.

Remarks

To count the number of barcode configurations within a barcode file saved on an RF device, first retrieve the file using the GetBarcodeFile Method, then use the BarcodeCount method.

BarcodeFileCount Method

This member of RFBarcode is supported on ALL devices.

The BarcodeFileCount method returns the total number of barcode files returned by the last successful call to the ListBarcodeFiles method.

VB

```
nBarcodes = object.BarcodeFileCount ()
```

VC++

```
HRESULT hr = object->BarcodeFileCount(short *nBarcodes);
```

Return Value

nBarcodes	The barcode file count variable.

Remarks

The BarcodeFileCount method will return the file count from the last successful call to the ListBarcodeFiles Method. Use the ListBarcodeFiles method to return the total number of barcode files that are currently stored on an RF device.

BarcodeFileName Method

This member of RFBarcode is supported on ALL devices.

The BarcodeFileName method returns the name of a barcode file on an RF device.

VB

```
pszName = object.BarcodeFileName (nBarcode)
```

VC++

```
HRESULT hr = object->BarcodeFileName(short nBarcode, Cstring
*pszName);
```

Return Value

pszName	The barcode file name variable. If this method returns an empty string, an error may have occurred. Use the RFGetLastError method to return the generated error code.

Parameters

nBarcode	The zero based index value of the barcode file from the RF device. For example, to return the file name of the third barcode file you would pass a 2.

Remarks

For a complete listing of all barcode files on an RF device, simply loop the BarcodeFileName method for the entire number of files returned by the BarcodeFileCount Method.

ClearBarcodes Method

This member of RFBarcode is supported on ALL devices.

The ClearBarcodes method clears the RFBarcode object of all barcode configurations.

VB

```
bStatus = object.ClearBarcodes ()
```

VC++

```
HRESULT hr = object->ClearBarcodes(BOOL *bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Remarks

To clear the barcode configurations from a saved barcode file, use the DeleteBarcodeFile Method.

Example

See the StoreBarcode Method.

Default Method

This member of RFBarcode is supported on ALL devices.

The Default method returns the default decode state for all barcode types except those explicitly defined within the current RFBarcode object.

VB

```
nDefault = object.Default()
```

VC++

```
HRESULT hr = object->Default(short *nDefault);
```

Return Value

nDefault	The default decode state variable (see Remarks). This method will return RF a -1 if an error occurs. Use the RFGetLastError to return the generated error message. The possible default decode states are as follows:

Remarks

The possible values are:

BCDISABLED	All barcodes not defined within the RFBarcode object will be disabled and will not be decoded for input.
BCENABLED	All barcodes not defined within the RFBarcode object will be enabled and will be decoded for input.
NO_DEFAULT	The default state for barcodes not defined within the RFBarcode will remain unchanged.

The default state for barcodes not defined within the RFBarcode object will be set using the StoreBarcode Method. Therefore, you must be sure that the current RFBarcode object has first been saved to an RF device before attempting to return the default decode state.

DeleteBarcodeFile Method

This member of RFBarcode is supported on ALL devices.

The DeleteBarcodeFile method removes barcode files from an RF device.

VB

```
bStatus = object.DeleteBarcodeFile (pszFileName)
```

VC++

```
HRESULT hr = object->DeleteBarcodeFile(LPCTSTR pszFileName,
BOOL *bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

pszFileName	The barcode file to be deleted. You may pass a single eight character file name or enter a wildcard symbol ("*") to delete all barcode files from an RF device.

Example

See the RFBarcode Samples.

GetBarcodeFile Method

This member of RFBarcode is supported on ALL devices.

The GetBarcodeFile method retrieves a barcode file from an RF device into the current RFBarcode object.

VB

```
nBarcodes = object.GetBarcodeFile (pszFileName)
```

VC++

```
HRESULT hr = object->GetBarcodeFile(LPCTSTR pszFileName,
short *nStatus);
```

Return Value

nBarcodes	The total number of barcode configurations placed within the current RFBarcode object. This method will return a -1 if an error occurs. Use the RFGetLastError method to return the error value. If the RFBarcode file does not exist on the RF device, this method will return 0.

Parameters

pszFileName	The file name for the RFBarcode file to be retrieved.

GetBarcodeType Method

This member of RFBarcode is supported on ALL devices.

The GetBarcodeType method returns the barcode type for a specific barcode configuration within the current RFBarcode object.

VB

```
nType = object.GetBarcodeType (nBarcode)
```

VC++

```
HRESULT hr = object->GetBarcodeType(short nBarcode, short
*nType);
```

Return Value

nType	The barcode type variable (see Remarks). This method will return a -1 if an error occurs. Use the RFGetLastError to return the generated error code.

Parameters

nBarcode	The zero based index value barcode configuration whose type is to be returned. The RFBarcode object has a zero based index. For example, to return the type for the third barcode configuration in the current RFBarcode object you would pass a 2.

Remarks

The possible values are:

CODE_39

UPC_A

UPC_E0

EAN_13

EAN_8

CODE_D25

CODE_I25

CODABAR

CODE_128

CODE_93

CODE_11

MSI

UPC_E1

WLNOSYMBOLOGY

See the Constant Values for the numeric equates returned by the function.

To return the type for a barcode configuration stored within a barcode file on an RF device, first use the GetBarcodeFile Method to set the barcode file as the current barcode object, then use GetBarcodeType.

GetDecode Method

This member of RFBarcode is supported on ALL devices.

The GetDecode method returns the decode state for a specific barcode configuration within the current RFBarcode object.

VB

```
nDecode = object.GetDecode (nBarcode)
```

VC++

```
HRESULT hr = object->GetDecode(short nBarcode, short
*nDecode);
```

Return Value

nDecode	The barcode configuration decode state variable (DECODEON or DECODEOFF). This method will return a -1 if an error occurs. Use the RFGetLastError to return the generated error code.

Parameters

nBarcode	The zero based index value of the specified barcode configuration. The RFBarcode object has a zero based index. For example, to return the decode state for the third barcode configuration in the current RFBarcode object you would pass a 2.

Remarks

The possible values are:

DECODEON	The barcode configuration will be decoded when scanned and is valid for input.
DECODEOFF	The barcode configuration will not be decoded when scanned and will not be accepted for input.

To return the decode state for a barcode configuration stored within a barcode file on an RF device, first use the GetBarcodeFile Method to set the barcode file as the current barcode object, then use GetDecode.

GetExpand Method

This member of RFBarcode is supported on ALL devices.

The GetExpand method returns the expand state for a barcode configuration within the current RFBarcode object array.

VB

```
bExpand = object.GetExpand (nBarcode)
```

VC++

```
HRESULT hr = object->GetExpand(short nBarcode, BOOL
*bExpand);
```

Return Value

bExpand	The current expand state for the barcode configuration (True or False).

Parameters

nBarcode	The zero-based index value of the barcode configuration that is being called. For example, to return the Expand state for the third barcode configuration in the current RFBarcode object you would pass a 2.

Remarks

To return the expand state for a barcode configuration stored within a barcode file on an RF device, first use the GetBarcodeFile Method to set the barcode file as the current barcode object, then use GetBarcodeType.

GetMaxLength Method

This member of RFBarcode is supported on ALL devices.

The GetMaxLength method returns the maximum input length for a specific barcode configuration within the current RFBarcode object.

VB

```
nLength = object.GetMaxLength (nBarcode)
```

VC++

```
HRESULT hr = object->GetMaxLength(short nBarcode, short
*nLength);
```

Return Value

nLength	The maximum input length variable. This method will return a -1 if an error occurs. Use the RFGetLastError method to return the generated error code.

Parameters

nBarcode	The zero based index value of the desired barcode configuration. For example, to return the maximum length for the third barcode configuration in the current RFBarcode object you would pass a 2.

Remarks

To return the maximum length for a barcode configuration stored within a barcode file on an RF device, first use the GetBarcodeFile Method to set the barcode file as the current barcode object, then use GetMaxLength.

GetMinLength Method

This member of RFBarcode is supported on ALL devices.

The GetMinLength method returns the minimum input length for a specific barcode configuration within the current RFBarcode object array.

VB

```
nMin = object.GetMinLength (nBarcode)
```

VC++

```
HRESULT hr = object->GetMinLength(short nBarcode, short
*nMin);
```

Return Value

nMin	The minimum input length variable. This function will return a -1 if an error occurs. Use the RFGetLastError method to return the generated error code.

Parameters

nBarcode	The zero based index value barcode configuration whose minimum input length is being returned. For example, to return the minimum length for the third barcode configuration in the current RFBarcode object you would pass a 2.

Remarks

To return the minimum length for a barcode configuration stored within a barcode file on an RF device, first use the GetBarcodeFile Method to set the barcode file as the current barcode object, then use GetMinLength.

ListBarcodeFiles Method

This member of RFBarcode is supported on ALL devices.

The ListBarcodeFiles method returns the total number of barcode files on an RF device and stores the list of file names within the current object.

VB

```
nBarcodes = object.ListBarcodeFiles()
```

VC++

```
HRESULT hr = object->ListBarcodeFiles(short *nBarcodes);
```

Return Value

nBarcodes	The total number of barcode files. This method will return a -1 if an error occurs. Use the RFGetLastError method to return the generated error code.

Remarks

To return the value of the last successful ListBarcodeFiles method without making an actual call to the RF device, use the BarcodeFileCount Method. To list the total number of barcode configurations within a single barcode file, use the BarcodeCount method after making the file the current RFBarcode object.

PullBarcode Method

This member of RFBarcode is supported on ALL devices.

The PullBarcode method restores a barcode file as the RF device's default barcode configuration while removing the barcode file from the RF device.

VB

```
bStatus = object.PullBarcode (pszFileName)
```

VC++

```
HRESULT hr = object->PullBarcode(LPCTSTR pszFileName, BOOL
*bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

pszFileName	The barcode file to be pulled.

Remarks

When used in conjunction, PushBarcode and PullBarcode allow you to temporarily change the settings of an RF device and return the device to its original settings for easy clean up.

Example

See the PushBarcode Method.

PushBarcode Method

This member of RFBarcode is supported on ALL devices.

The PushBarcode method stores the RF device's default barcode configuration to a file for later restoration.

VB

```
bStatus = object.PushBarcode (pszFileName)
```

VC++

```
HRESULT hr = object->PushBarcode(LPCTSTR pszFileName, BOOL
*bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

pszFileName	The file name for the RF device's default barcode configuration.

Remarks

Use the PullBarcode Method to restore a previously saved barcode file while removing the file from the RF device's memory. When used in conjunction, PushBarcode and PullBarcode allow you to temporarily change the settings of an RF device and return the device to its original settings for easy clean up.

Example

```
' VB Sample Code
Dim wlBCode As New RFBarcode
.
.
.
  bStatus = wlBCode.PushBarcode("Defltbar")
' create, store, and use barcode configurations in the app
.
.
.
  ' use PullBarcode to restore the default barcode at
  ' cleanup
  bStatus = wlBCode.PullBarcode ("Defltbar")
```

RemoveBarcode Method

This member of RFBarcode is supported on ALL devices.

The RemoveBarcode method removes a specific barcode configuration from the current RFBarcode object.

VB

```
bStatus = object.RemoveBarcode (nBarcode)
```

VC++

```
HRESULT hr = object->RemoveBarcode(short nBarcode, BOOL
*bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

nBarcode	The zero based index value barcode configuration to be removed. For example, to remove the third barcode configuration in the current RFBarcode object you would pass a 2.

Remarks

To remove a barcode configuration from a stored barcode file on an RF device, you must first set the barcode file as the current RFBarcode object using the GetBarcodeFile Method, then use RemoveBarcode.

RFGetLastError Method

This member of RFBarcode is supported on ALL devices.

The RFGetLastError method returns the value of the last generated error.

VB

```
dError = object.RFGetLastError()
```

VC++

```
HRESULT hr = object->RFGetLastError(DWORD *dError);
```

Return Value

dError	The numeric DWORD value of the last generated function error.

Remarks

Whenever a function error occurs, an error value is generated. You may use RFGetLastError to return the value of the error. The generated errors are as follows:

WLNOERROR	No error returned from function call.
WLCOMMERROR	RF network communication error
WLMEMORYERROR	Memory allocation error
WLNOTINITIALIZED	RF object is improperly initialized.
WLREQUESTFAIL	Input request string format failure
WLINVALIDPARAMS	Invalid parameter(s) for function call
WLINVALIDRETURN	Invalid return designator for function call
WLFUNCTIONFAILED	RF device returned a failure code for the function
WLBCPARSEERROR	Failure parsing barcode file
WLBCADDERROR	Failed to add barcode to object
WLBCCLEARERROR	Failed to clear barcode list
WLBARCODELIMITEXCEEDED	Too many barcodes for configuration file
WLTONEADDERROR	Failed to add tone to object
WLTONECLEARERROR	Failed to clear tone list

WLIOQUEUEERROR	IO queue is empty
WLNOINPUTTYPE	Unable to determine input type
WLPORTTIMEOUT	Auxiliary port timeout
WLDTOUTOFSYNC	Terminal date/time out of sync by more than five seconds

See the Constant Values for the numeric equates returned by the function.

Example

See the RFBarcode Samples.

SetBarcodeType Method

This member of RFBarcode is supported on ALL devices.

The SetBarcodeType method defines the barcode type for an existing barcode configuration within an RFBarcode object.

VB

```
bStatus = object.SetBarcodeType (nType, nBarcode)
```

VC++

```
HRESULT hr = object->SetBarcodeType(short nType, short
nBarcode, BOOL *bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

nType	The barcode type (see Remarks).
nBarcode	The zero based index value barcode configuration whose type is to be altered. For example, to alter the type for the third barcode configuration in the current RFBarcode object you would pass a 2.

Remarks

The possible values for nType are:

CODE_39

UPC_A

UPC_E0

EAN_13

EAN_8

CODE_D25

CODE_I25

CODABAR

CODE_128

CODE_93

CODE_11

MSI

UPC_E1

WLNOSYMBOLOGY

To alter the type for a barcode configuration stored within a barcode file on an RF device, first use the GetBarcodeFile Method to set the barcode file as the current RFBarcode object within the application, then use SetBarcodeType.

SetDecode Method

This member of RFBarcode is supported on ALL devices.

The SetDecode method defines the decode state for an existing barcode configuration within an RFBarcode object.

VB

```
bStatus = object.SetDecode (nDecode, nBarcode)
```

VC++

```
HRESULT hr = object->SetDecode(short nDecode, short
nBarcode, BOOL *bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

nDecode	The decode state for the barcode configuration, as represented by the following constants (DECODEON or DECODEOFF).
nBarcode	The zero based index value barcode configuration whose decode state is to be altered. For example, to alter the decode state for the third barcode configuration in the current RFBarcode object you would pass a 2.

Remarks

The possible values for nDecode are:

DECODEON	The barcode configuration will be decoded when scanned and is valid for input.
DECODEOFF	The barcode configuration will not be decoded when scanned and will not be accepted for input.

To alter the decode state for a barcode configuration stored within a barcode file on an RF device, first use the GetBarcodeFile Method to set the barcode file as the current barcode object within the application, then use SetDecode.

SetExpand Method

This member of RFBarcode is supported on ALL devices.

The SetExpand method defines the expand state for an existing barcode configuration within an RFBarcode object.

VB

```
bStatus = object.SetExpand (bExpand, nBarcode)
```

VC++

```
HRESULT hr = object->SetBarcode(BOOL bExpand, short
nBarcode, BOOL *bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

bExpand	The expand state for the existing barcode configuration (True or False).
nBarcode	The zero based index value of the specified barcode configuration. For example, to alter the expand state for the third barcode configuration in the current RFBarcode object you would pass a 2.

Remarks

To alter the expand state for a barcode configuration stored within a barcode file on an RF device, first use the GetBarcodeFile Method to set the barcode file as the current barcode object within the application, then use SetExpand.

SetMaxLength Method

This member of RFBarcode is supported on ALL devices.

The SetMaxLength method defines the maximum input length for an existing barcode configuration within an RFBarcode object.

VB

```
bStatus = object.SetMaxLength (nMax, nBarcode)
```

VC++

```
HRESULT hr = object->SetMaxLength(short nMax, short
nBarcode, BOOL *bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

nMax	The maximum acceptable input length for the barcode configuration.
nBarcode	The zero based index value barcode configuration whose maximum input length is to be altered. For example, to alter the maximum input length for the third barcode configuration in the current RFBarcode object you would pass a 2.

Remarks

Input length restrictions for barcode configurations can be eliminated by setting the *nMax* parameter of the SetMaxLength and the *nMin* parameter of the SetMinLength Method to zero(0).

To alter the maximum input length for a barcode configuration stored within a barcode file on an RF device, first use the GetBarcodeFile Method to set the barcode file as the current barcode object within the application, then use SetMaxLength.

SetMinLength Method

This member of RFBarcode is supported on ALL devices.

The SetMinLength method defines the minimum input length for an existing barcode configuration within an RFBarcode object.

VB

```
bStatus = object.SetMinLength (nMin, nBarcode)
```

VC++

```
HRESULT hr = object->SetMinLength(short nMin, short
nBarcode, BOOL *bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

nMin	The minimum input length for the barcode configuration.
nBarcode	The zero based index value barcode configuration whose minimum input length is to be altered. For example, to alter the minimum input length for the third barcode configuration in the current RFBarcode object you would pass a 2.

Remarks

To alter the minimum input length for a barcode configuration stored within a barcode file on an RF device, first use the GetBarcodeFile Method to set the barcode file as the current barcode object within the application, then use SetMinLength.

StoreBarcode Method

This member of RFBarcode is supported on ALL devices.

The StoreBarcode method saves the current RFBarcode object as a barcode file on an RF device.

VB

```
bStatus = object.StoreBarcode (pszFileName, nDefault)
```

VC++

```
HRESULT hr = object->StoreBarcode(LPCTSTR pszFileName, short
nDefault, BOOL *bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

pszFileName	The file name that the current RFBarcode object will be saved as on the RF device. This file name may be up to 8 characters in length.
nDefault	The default action taken with the barcodes NOT defined by the current configuration (see Remarks).

Remarks

The possible values are:

BCDISABLED	All barcodes not defined within the RFBarcode object will be disabled and will not be decoded for input.
BCENABLED	All barcodes not defined within the RFBarcode object will be enabled and will be decoded for input.
NO_DEFAULT	The default state for barcodes not defined within the RFBarcode will remain unchanged.

To use the current RFBarcode object with an RFInput call, you must first save the object to the RF device as a barcode file using the StoreBarcode method.

Example

```
' VB Sample Code
Dim wlBCode As New RFBarcode
.
.
.
  bStatus = wlBCode.PushBarcode("Defltbar")
  wlBCode.ClearBarcodes
  wlBCode.AddBarcode CODE_39, False, DECODEON, 0, 0
  wlBCode.AddBarcode UPC_A, False, DECODEON, 12, 12
  If wlBCode.StoreBarcode ("CycCtBar", BCENABLED) = False Then
    GoTo ExitApp
  End If
  ' you can use the CycCtBar barcode configuration in an input
  ' call such as RFInput
```

RFBarcode Samples

Using both Visual Basic and Visual C++ sample code , the following set of applications will demonstrate the use of the RFBarcode object and its methods. Both applications will save two barcode configurations to the RF device, limit the barcodes you may scan based upon the barcode configurations, then return the RF device to its original state when finished.

Visual Basic Sample:

```
' RFBarcode Object Demo Application
'
' Import a sleep function from the kernel32 dll
Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)
' Declare an RFIO object for use throughout the
' demo application.
Public rfConsole As New RFIO
' Declare an RFBarcode object for use throughout the
' demo application.
Public rfBCode As New RFBARCODE
' strBuffer is a temp storage buffer
Dim strBuffer As String
' bStored is the main screen's storage flag
Dim bStored As Boolean

Public Sub Main()
' This application demonstrates the use of the various
' RFBarcode methods. It assumes a display area of 20x8.
' In practice, one should use an RFTerminal
' object to format output to fit the current device.
' The following lines clear the display and output the
' app title in reverse video.
    rfConsole.RFPrint 0, 0, "   Welcome To The   ", _
              WLCLEAR + WLREVERSE
    rfConsole.RFPrint 0, 1, "   RFBarcode Demo   ", WLREVERSE
    rfConsole.RFPrint 0, 3, "Hit a key to proceed", WLNORMAL
' Hide the cursor for a cleaner display
    rfConsole.RFPrint 0, 9, "", WLNORMAL
'
' Flush the output buffer and await a keystroke/scan
    If rfConsole.GetEvent() = "" Then
        GoTo ExitApp
    End If  ' If rfConsole.GetEvent() = ""

' When writing RF applications, the ability to determine
' which barcodes may be scanned is of great importance.
' If the application can limit its input, there
```

```
' is less chance of incorrect data corrupting your database.
' For demonstration purposes, we will limit our input to two
' symbologies: CODE_39, which will represent a bin location,
' & UPC_A, which will represent an item.
    rfBCode.AddBarcode CODE_39, False, DECODEON, 0, 0    ' Scan
                                ' any CODE_39 barcode
    rfBCode.AddBarcode UPC_A, False, DECODEON, 12, 12    ' Scan
                                ' UPC_A barcode of 12
                                 'length

' Store the current barcode configuration on the RF device. We will
' disable all other barcode types to prevent misscans.
    If rfBCode.StoreBarcode("BCDemo", BCDISABLED) = False Then
        rfConsole.RFPrint 0, 2, "StoreBarcode Failed!", _
                  WLCLEAR + WLREVERSE
        rfConsole.GetEvent
        GoTo ExitApp
    End If   ' If rfBCode.StoreBarcode("BCDemo", BCDISABLED) = False

    bStored = False
    While True
        If bStored = False Then
            rfConsole.RFPrint 0, 0, " Pseudo-Cycle Count ", _
                      WLCLEAR + WLREVERSE
            rfConsole.RFPrint 0, 2, "Input Bin/Item : ", WLNORMAL
            rfConsole.RFPrint 0, 7, "  CLR To Exit Demo  ", _
                      WLREVERSE + WLFLUSHOUTPUT
            bStored = rfConsole.PushScreen("PCCMain")
        Else: rfConsole.RestoreScreen "PCCMain"
        End If  ' If bStored = False

        strBuffer = rfConsole.RFInput("", 20, 0, 3, "BCDemo", _
                              NORMALKEYS, _
                              WLBACKLIGHT + WLDISABLE_KEY)
        If rfConsole.RFGetLastError() <> WLNOERROR Then
            GoTo ExitApp
        End If  ' If rfConsole.RFGetLastError() <> WLNOERROR

        If rfConsole.LastInputType() = WLCOMMANDTYPE And _
                                       Asc(strBuffer) = 27 Then
            GoTo ExitApp
        Else
            If rfConsole.LastBarcodeType() = CODE_39 Then
            ' In a real cycle count application, you would be
            ' validating the new bin location for
            ' the item counts.
```

```
                    rfConsole.RFPrint 0, 2, "Bin Number Scanned", _
                              WLCLREOLN
                    rfConsole.RFPrint 0, 3, strBuffer, WLCLREOLN
                ElseIf rfConsole.LastBarcodeType() = UPC_A Then


                ' Here you would request an item count or add one
                ' to the database record for the item
                ' scanned.
                    rfConsole.RFPrint 0, 2, "Item Number Scanned", _
                              WLCLREOLN
                    rfConsole.RFPrint 0, 3, strBuffer, WLCLREOLN
                End If   ' If rfConsole.LastBarcodeType() = CODE_39

                rfConsole.RFPrint 0, 7, "Hit a key to proceed", WLNORMAL
                ' Hide the cursor for a cleaner display
                rfConsole.RFPrint 0, 9, "", WLNORMAL

                ' Flush the output buffer and await a keystroke/scan
                If rfConsole.GetEvent() = "" Then
                    GoTo ExitApp
                End If  ' If rfConsole.GetEvent() = ""
            End If  ' If rfConsole.LastInputType() = WLCOMMANDTYPE

        Wend    ' While True

' Anytime an application changes the configuration of an RF device,
' it should return it to its original values during cleanup
    rfBCode.DeleteBarcodeFile "BCDemo"

EitApp:
End Sub
```

Visual C++ Sample:

```
////////////////////////////////////////////////////////
//     Place RF application's main code here
IRFIO           rfConsole;          // Declare an RFIO interface
                                    // object
IRFBarcode      rfBCode;            // Declare an RFBarcode
                                    // interface object
Cstring         csBuffer;           // Storage buffer for data
                                    // strings
Short           nTmp;               // Temporary short variable
BOOL            bDone = FALSE;      // Loop Control Variable
BOOL            bStored = FALSE;    // Main screen stored flag
    // Create an RFIO object in the automated server for our use
    if (rfConsole.CreateDispatch ("WAVELINKOLE.RFIO") == FALSE) ↵
         return FALSE;
    // Create an RFBarcode object in the automated server
    // for our use
    if (rfBCode.CreateDispatch ("WAVELINKOLE.RFBARCODE") == FALSE) {
    // Release the RFIO object
    rfConsole.ReleaseDispatch ();
         return FALSE;
    }    // if (rfBCode.CreateDispatch ("WAVELINKOLE.RFBARCODE") ==
         // FALSE)

    // This application demonstrates the use of the various
    // RFBarcode methods.
    // It assumes a display area of 20x8. In practice, one should
    // use an RFTerminal object to format output to fit the current
    // device.
    // The following lines clear the display and output the
    // app title in reverse video.
    rfConsole.RFPrint (0,0, "   Welcome To The   ", ↵
              WLCLEAR | WLREVERSE);
    rfConsole.RFPrint (0, 1, "   RFBarcode Demo   ", WLREVERSE);
    rfConsole.RFPrint (0, 3, "Hit a key to proceed", WLNORMAL);
    // Hide the cursor for a cleaner display
    rfConsole.RFPrint (0, 9, "", WLNORMAL);

    // Flush the output buffer and await a keystroke/scan
    if ((rfConsole.GetEvent ()).IsEmpty () == TRUE) {
        // Release the RFIO & RFBarcode objects
        rfConsole.ReleaseDispatch ();
        rfBCode.ReleaseDispatch ();
        return FALSE;
    } // if ((rfConsole.GetEvent ()).IsEmpty () == TRUE)
```

```
    // When writing RF applications, the ability to determine which
    // barcodes may be scanned is of great importance. If the
    // application can limit its input, there is less chance of
    // incorrect data corrupting your database.
    // For demonstration purposes, we will limit our input to two
    // symbologies:
    // CODE_39, which will represent a bin location, & UPC_A,
    // which will represent an item.
    rfBCode.AddBarcode (CODE_39, FALSE, DECODEON, 0, 0); // Scan
                                                         // any CODE_39
                                                         // barcode
    rfBCode.AddBarcode (UPC_A, FALSE, DECODEON, 12, 12); // Scan
                                                         // UPC_A barcode
                                                         // of 12 length

    // Store the current barcode configuration on the RF device.
    // We will disable all other barcode types to prevent misscans.
    if (rfBCode.StoreBarcode ("BCDemo", BCDISABLED) == FALSE) {
        rfConsole.RFPrint (0, 2, "StoreBarcode Failed!", ↵
                  WLCLEAR | WLREVERSE);
        rfConsole.GetEvent ();
        // Release the RFIO & RFBarcode objects
        rfConsole.ReleaseDispatch ();
        rfBCode.ReleaseDispatch ();
        return FALSE;
    }   // if (rfBCode.StoreBarcode ("BCDemo", BCDISABLED) == FALSE)
// NOTE : It is a safer practice to use a const string for the
// barcode filename to prevent typos (ex. const char* const BCNAME
// = "BCDemo";).

    while (bDone == FALSE) {
        if (bStored == FALSE) {
            rfConsole.RFPrint (0, 0, " Pseudo-Cycle Count ", ↵
                      WLCLEAR | WLREVERSE);
            rfConsole.RFPrint (0, 2, "Input Bin/Item : ", WLNORMAL);
            rfConsole.RFPrint (0, 7, "  CLR To Exit Demo  ", ↵
                      WLREVERSE | WLFLUSHOUTPUT);
            bStored = rfConsole.PushScreen ("PCCMain");
        }     // if (bStored == FALSE)
        else rfConsole.RestoreScreen ("PCCMain");

        csBuffer = rfConsole.RFInput (NULL, 20, 0, 3, "BCDemo", ↵
                   NORMALKEYS,WLBACKLIGHT | WLDISABLE_KEY);

        if (rfConsole.LastInputType () == WLCOMMANDTYPE &&
            csBuffer [0] == char(27))
```

```
            bDone = TRUE;
        else {
            if (rfConsole.LastBarcodeType () == CODE_39) {
            // In a real cycle count application, you would be
            // validating the new location for the item counts.
                   rfConsole.RFPrint (0, 2, "Bin Number Scanned", ↵
                            WLCLREOLN);
                   rfConsole.RFPrint (0, 3, csBuffer, WLCLREOLN);
            } // if (rfConsole.LastBarcodeType () == CODE_39)
            else if (rfConsole.LastBarcodeType () == UPC_A) {
            // Here you would request an item count or add one
            // to the database record for the item scanned.
                    rfConsole.RFPrint (0, 2,  ↵
                             "Item Number Scanned", WLCLREOLN);
                   rfConsole.RFPrint (0, 3, csBuffer, WLCLREOLN);
            }
            // else if (rfConsole.LastBarcodeType () == UPC_A)

            rfConsole.RFPrint (0, 7, "Hit a key to proceed", ↵
                      WLNORMAL);
            // Hide the cursor for a cleaner display
            rfConsole.RFPrint (0, 9, "", WLNORMAL);
            // Flush the output buffer and await a keystroke/scan
            if ((rfConsole.GetEvent ()).IsEmpty () == TRUE) {
                   // Release the RFIO & RFBarcode objects
                   rfConsole.ReleaseDispatch ();
                   rfBCode.ReleaseDispatch ();
                   return FALSE;
            }
            // if ((rfConsole.GetEvent ()).IsEmpty () == TRUE)

        } // else

    } // while (bDone == FALSE)

    // Anytime an application changes the configuration of an
    // RF device, it should return it to its original values
    // during cleanup
    rfBCode.DeleteBarcodeFile ("BCDemo");

    // Release the RFIO & RFBarcode objects
    rfConsole.ReleaseDispatch ();
    rfBCode.ReleaseDispatch ();
```

RFError Object

The RFError object is supported on ALL devices.

The RFError object lets you define and display error messages within your applications.

Methods

ClearError SetErrorLine
Display

Prog IDs

"WAVELINKOLE.RFERROR"

Remarks

Once defined, an error message will remain within the RFError object until either the object is released, a new message is defined over an existing line of error message using SetErrorLine, or all message lines are cleared using ClearError. This allows you to use a single error object within your application which may be modified for each error message rather than creating a new error object for each individual error message.

Method Summary

ClearError Method
- Clears error messages from the RFError object.

Display Method
- Displays error messages.

SetErrorLine Method
- Defines error message text at a specific row on the device.

ClearError Method

This member of RFError is supported on ALL devices.

The ClearError method allows you to clear the RFError object of any error messages.

VB

```
bStatus = object.ClearError ()
```

VC++

```
HRESULT hr = object->ClearError(BOOL *bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values.

Remarks

Once defined, an error message will remain within an RFError object until either the object is released, a new message is defined over an existing line of the error message using SetErrorLine, or all message lines are cleared using ClearError. This allows you to use a single error object within your application that may be modified for each error message rather than creating a new error object for each individual error message.

Example

See the Display Method.

Display Method

This member of RFError is supported on ALL devices.

The Display method allows you to display an error message, as defined by SetErrorLine , on a remote RF device.

VB

```
bStatus = object.Display (nTimeout)
```

VC++

```
HRESULT hr = object->Display(short nTimeout, BOOL *bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values.

Parameters

nTimeout	A numeric value representing in seconds the total display time for the Error message, as defined by the SetErrorLine method. If this value is set to zero (0), the error message will remain displayed until the user responds to a prompt.

Example

```
' VB Sample Code
Dim wlerror As New RFError
.
.
.
  wlerror.ClearError
  wlerror.SetErrorLine "  Please Have", 0
  wlerror.SetErrorLine " Customer Sign", 1
  wlerror.SetErrorLine "Work Order Sheet", 2
  wlerror.Display 0
```

SetErrorLine Method

This member of RFError is supported on ALL devices.

The SetErrorLine method lets you define the specific text and display row for an error message.

VB

```
bStatus = object.SetErrorLine (pszText, nRow)
```

VC++

```
HRESULT hr = object->SetErrorLine(LPCTSTR pszText, short
nRow, BOOL *bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values.

Parameters

pszText	The text for the line of an error message.
nRow	The device row in which the error message should appear. The first device row is zero (0).

Remarks

Once defined, an error message will remain within an RFError object until either the object is released, a new message is defined over an existing line of error message using SetErrorLine, or all message lines are cleared using ClearError. This allows you to use a single error object within your application that may be modified for each error message rather than creating a new error object for each individual error message.

Example

See the Display Method.

RFError Samples

Using both Visual Basic and Visual C++ sample code, the following set of applications demonstrate the use of the RFError object and its methods.

Visual Basic Sample:

```
' RFError Object Demo Application
'
' Import a sleep function from the kernel32 dll
Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)
' Declare an RFIO object for use throughout the
' demo application.
Public rfConsole As New RFIO
' Declare an RFError object for use throughout the
' demo application.
Public rfErr As New RFERROR
' strBuffer is a temp storage buffer
Dim strBuffer As String

Dim nLcv As Integer

Public Sub Main()

' This application demonstrates the use of the various RFError
' methods. It assumes a display area of 20x8. In practice, one
' should use an RFTerminal object to format output to fit the
' current device.
' The following lines clear the display and output the app title in
'  reverse video.
    rfConsole.RFPrint 0, 0, "  Welcome To The    ", _
              WLCLEAR + WLREVERSE
    rfConsole.RFPrint 0, 1, "RFError Method Demo ", WLREVERSE
    rfConsole.RFPrint 0, 3, "Hit a key to proceed", WLNORMAL
' Hide the cursor for a cleaner display
    rfConsole.RFPrint 0, 9, "", WLNORMAL
'
' Flush the output buffer and await a keystroke/scan
    If rfConsole.GetEvent() = "" Then
        GoTo ExitApp
    End If  ' If rfConsole.GetEvent() = "" Then
' During the course of any application, there will be times
' when you wish to notify a user of an error condition which has
' arisen. The RFError object is the preferred method for doing
' this over an RF network.
'
' The first sample will define and display an error message for 2
```

```
' seconds
    rfErr.SetErrorLine " This error will ", 0
    rfErr.SetErrorLine "  display for 2  ", 1
    rfErr.SetErrorLine "     seconds     ", 2
'
    If rfErr.Display(2) = False Then
        GoTo ExitApp
    End If  ' if rfErr.Display (2) = False
'
' Now we will define and display an error message, but require the
' user to acknowledge the error by pressing the CLR (Escape) key by
' setting a zero (0) timeout.
    rfErr.ClearError
    rfErr.SetErrorLine "Acknowledge this", 1
    rfErr.SetErrorLine " error, please. ", 2
    rfErr.SetErrorLine " ", 3

    If rfErr.Display(0) = False Then
        GoTo ExitApp
    End If  ' if rfErr.Display (0) = False

' When an application must perform an extended operation, it is
' usually a good idea to provide the user with a visual indicator
' of progress.
    rfConsole.RFPrint 0, 0, " Preforming a time- ", WLCLEAR
    rfConsole.RFPrint 0, 1, "   consuming job!   ", WLFLUSHOUTPUT
'
    rfErr.ClearError
    For nLcv = 10 To 100 Step 10
        strBuffer = " " + Format(nLcv / 100, "0.00%") + " Complete "
        rfErr.SetErrorLine strBuffer, 0
        rfErr.SetErrorLine " <CLR> To Cancel. ", 1
        If rfErr.Display(1) = False Then
            GoTo ExitApp
        End If  ' if rfError.Display (1) = False
        If rfConsole.TellEvent() <> "" Then
            strBuffer = rfConsole.GetEvent()
            If Asc(strBuffer) = 27 Then
                GoTo ExitApp
            End If  'If Asc(strBuffer) = 27
        End If  ' If rfConsole.TellEvent() <> "" Then
        Call Sleep(2000)
    Next nLcv

ExitApp:
End Sub
```

Visual C++ Example

```
//////////////////////////////////////////////////////
//          Place RF application's main code here
IRFIO       rfConsole;       // Declare an RFIO interface object
IRFError    rfError;         // Declare an RFError interface object
Cstring     csBuffer;        // Storage buffer for data strings
    // Create an RFIO object in the automated server for our use
    if (rfConsole.CreateDispatch ("WAVELINKOLE.RFIO") == FALSE) ↵
        return FALSE;
    // Create an RFError object in the automated server for our use
    if (rfError.CreateDispatch ("WAVELINKOLE.RFERROR") == FALSE) {
        // Release the RFIO object
        rfConsole.ReleaseDispatch ();
        return FALSE;
    } // if (rfError.CreateDispatch ("WAVELINKOLE.RFERROR") ↵
                    == FALSE)

    // This application demonstrates the use of the various RFError
    // methods. It assumes a display area of 20x8. In practice,
    // one should use an RFTerminal object to format output to fit
    // the current device.
    // The following lines clear the display and output the app
    // title in reverse video.
    rfConsole.RFPrint (0, 0, "  Welcome To The    ",
             WLCLEAR | WLREVERSE);
    rfConsole.RFPrint (0, 1, "RFError Method Demo ", WLREVERSE);
    rfConsole.RFPrint (0, 3, "Hit a key to proceed", WLNORMAL);
    // Hide the cursor for a cleaner display
    rfConsole.RFPrint (0, 9, "", WLNORMAL);

    // Flush the output buffer and await a keystroke/scan
    if ((rfConsole.GetEvent ()).IsEmpty () == TRUE) {
        // Release the RFIO & RFError objects
        rfConsole.ReleaseDispatch ();
        rfError.ReleaseDispatch ();
        return FALSE;
    } // if ((rfConsole.GetEvent ()).IsEmpty () == TRUE)

    // During the course of any application, there will be times
    // when you wish to notify a user of an error condition which
    // has arisen. The RFError object is the preferred method for
    // doing this over an RF network.

    // The first sample will define and display an error message
```

```
// for 2 seconds
rfError.SetErrorLine (" This error will ", 0);
rfError.SetErrorLine ("  display for 2  ", 1);
rfError.SetErrorLine ("     seconds     ", 2);

if (rfError.Display (2) == FALSE) {
    // Release the RFIO & RFError objects
    rfConsole.ReleaseDispatch ();
    rfError.ReleaseDispatch ();
    return FALSE;
} // if (rfError.Display (2) == FALSE)

// Now we will define and display an error message, but
// require the user to acknowledge the error by pressing
// the CLR (Escape) key by setting a zero (0) timeout.
rfError.ClearError ();
rfError.SetErrorLine ("Acknowledge this", 1);
rfError.SetErrorLine (" error, please. ", 2);
rfError.SetErrorLine (" ", 3);

if (rfError.Display (0) == FALSE) {
    // Release the RFIO & RFError objects
    rfConsole.ReleaseDispatch ();
    rfError.ReleaseDispatch ();
    return FALSE;
} // if (rfError.Display (0) == FALSE)

// When an application must perform an extended operation,
// it is usually a good idea to provide the user with a
// visual indicator of progress.
rfConsole.RFPrint (0, 0, " Preforming a time- ", WLCLEAR);
rfConsole.RFPrint (0, 1, "   consuming job!   ", WLFLUSHOUTPUT);

rfError.ClearError ();
for (int nLcv = 0; nLcv <= 100; nLcv += 10) {
    csBuffer.Format ("  %%%03d Complete  ", nLcv);
    rfError.SetErrorLine (csBuffer, 0);
    rfError.SetErrorLine (" <CLR> To Cancel. ", 1);
    if (rfError.Display (1) == FALSE) {
        // Release the RFIO & RFError objects
        rfConsole.ReleaseDispatch ();
        rfError.ReleaseDispatch ();
        return FALSE;
    } // if (rfError.Display (1) == FALSE)
    if ((rfConsole.TellEvent ()).IsEmpty () == FALSE) {
        csBuffer = rfConsole.GetEvent ();
```

```
            if (csBuffer [0] == char(27)) break;
        } // if (rfConsole.TellEvent ()).IsEmpty () == FALSE)
        Sleep (2000);
    } // for (int nLcv = 0; nLcv <= 100; nLcv += 10)

    // Release the RFIO & RFError objects
    rfConsole.ReleaseDispatch ();
    rfError.ReleaseDispatch ();
```

RFFile Object

The RFFile object is supported on ALL devices.

The RFFile object provides basic file IO capabilities for remote RF devices. Using an RFFile object you may save and delete DOS based files to an RF Device over a WaveLink wireless network.

Methods

RFDeleteFile	RFGetFile
RFFileCount	RFGetLastError
RFFileDate	RFListFiles
RFFileName	RFListFilesEx
RFFileSize	RFStoreFile
RFFileTime	RFTransferFile

Prog IDs

"WAVELINKOLE.RFFILE"

Remarks

The RFFile object uses the standard DOS 8.3 file and wild card ("*") naming convention.

Method Summary

RFDeleteFile Method
- Deletes DOS files from the RF device.

RFFileCount Method
- Returns the total number of files returned by the last call to the RFListFiles method.

RFFileDate Method
- Returns the last modification date of a specific file.

RFFileName Method
- Returns the name of a specific file

RFFileSize Method
- Returns the size of a specific file.

RFFileTime Method
- Returns the last modification time of a specific file.

RFGetFile Method
- Retrieves a specific file into the current RFFile object.

RFGetLastError Method
- Returns the value of the last generated RFFile error.

RFListFiles Method
- Returns the number of files on the device and stores the file names in the current RFFile object.

RFListFilesEx Method
- Returns the number of files on the device and stores the file names in the current RFFile object. This method also stores the size, the last modification date, and the last modification time of the file.

RFStoreFile Method
- Stores data as a DOS file on the RF device.

RFTransferFile Method
- Transfers files between a network host and the RF device.

RFDeleteFile Method

This member of RFFile is supported on ALL devices.

The RFDeleteFile method removes DOS files from an RF device.

VB

```
bStatus = object.RFDeleteFile (pszFileName)
```

VC++

```
HRESULT hr = object->RFDeleteFile(LPCTSTR pszFileName, BOOL
*bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

pszFileName	The DOS file to delete from the RF device.

Remarks

The RFFile object uses the standard DOS 8.3 file and wild card ("*") naming convention. For example, to delete all files from an RF devices memory, pass "*.*" for the pszFileName parameter.

Example

See the RFFile Samples.

RFFileCount Method

This member of RFFile is supported on ALL devices.

The RFFileCount method returns the total number of files returned by the last successful call to the RFListFiles method.

VB

```
nFiles = object.RFFileCount ()
```

VC++

```
HRESULT hr = object->RFFileCount(short *nFiles);
```

Return Value

nFiles	The number of files returned by the last call to RFListFiles.

Remarks

The RFFileCount method will return the file count from the last successful call to theRFListFiles Method. Use the RFListFiles method to return the total number of files that are currently stored on an RF device.

Example

See the RFFile Samples.

RFFileDate Method

This member of RFFile is supported on ALL devices.

The RFFileDate method returns the last file modification date of the file in the position specified by *nFile*.

VB

```
pszDate = object.RFFileDate (nFile)
```

VC++

```
HRESULT hr = object->RFFileDate(short nFile, LPCTSTR
*pszDate);
```

Return Value

pszDate	The date the file was last modified. The date will be returned in the format "YYYYMMDD. If this method returns an empty string, an error may have occurred. Use the RFGetLastError method to return the generated error code.

Parameters

nFile	The zero-based index value file on the RF device that's last modification date is being returned. For example, to return the modification date of the third file you would pass a 2.

Remarks

You must use the RFListFilesEx Method rather than the RFListFiles method to store the additional file time information within the RFFile object.

Use the RFFileTime Method to return the time a file was last modified. Use the RFFileSize Method to return the size of file.

RFFileName Method

This member of RFFile is supported on ALL devices.

The RFFileName method returns the name of a file saved on an RF device.

VB

```
pszName = object.RFFileName (nFile)
```

VC++

```
HRESULT hr = object->RFFileName(short nFile, Cstring
*pszName);
```

Return Value

pszName	The file name variable. If this method returns an empty string, an error may have occurred. Use the RFGetLastError method to return the generated error code.

Parameters

nFile	The zero based index value file on the RF device that's file name is being returned. For example, to return the file name of the third file you would pass a 2.

Remarks

For a complete listing of all files on an RF device, first retrieve the list using the RFListFiles Method , then loop the RFFileName method for the entire number of files returned by the RFFileCount Method.

RFFileSize Method

This member of RFFile is supported on ALL devices.

The RFFileSize method returns the exact size of the file in the position specified by *nFile*.

VB

```
lSize = object.RFFileSize (nFile)
```

VC++

```
HRESULT hr = object->RFFileSize(short nFile, long *lSize);
```

Return Value

lSize	The size of the file in bytes. If this method returns an empty string, an error may have occurred. Use the RFGetLastError method to return the generated error code.

Parameters

nFile	The zero based index value file on the RF device that's size is being returned. For example, to return the size of the third file you would pass a 2.

Remarks

You must use the RFListFilesEx Method rather than the RFListFiles Method to store the additional file size information within the RFFile object. Use the RFFileDate Method to return the date a file was last modified. Use the RFFileTime Method to return the time a file was last modified.

RFFileTime Method

This member of RFFile is supported on ALL devices.

The RFFileTime method returns the last file modification time of the file in the position specified by *nFile*.

VB

```
pszTime = object.RFFileTime (nFile)
```

VC++

```
RFFileTime(short nFile, LPCTSTR *pszTime);
```

Return Value

pszTime	The time the file was last modified. The time will be returned in the format "HHMMSS. If this method returns an empty string, an error may have occurred. Use the RFGetLastError method to return the generated error code.

Parameters

nFile	The zero based index value file on the RF device that's last modification time is being returned. For example, to return the modification time of the third file you would pass a 2.

Remarks

You must use the RFListFilesEx Method rather than the RFListFiles Method to store the additional file time information within the RFFile object. Use the RFFileDate Method to return the date a file was last modified. Use the RFFileSize Method to return the size of the file.

RFGetFile Method

This member of RFFile is supported on ALL devices.

The RFGetFile method retrieves a file from an RF device into the current RFFile object.

VB

```
pszFileData = object.RFGetFile (pszFileName)
```

VC++

```
HRESULT hr = object->RFGetFile(LPCTSTR pszFileName, Cstring
*pszFileData);
```

Return Value

pszFileData	The data stored in the file designated by pszFileName. If pszFileData is empty, use RFGetLastError method to return the error value.

Parameters

pszFileName	The file to be set as the current RFFile object.

Remarks

The RFGetFile object uses the standard DOS 8.3 file naming convention.

RFGetLastError Method

This member of RFFile is supported on ALL devices.

The RFGetLastError method returns the value of the last generated error.

VB

```
dError = object.RFGetLastError()
```

VC++

```
HRESULT hr = object->GetLastError(DWORD *dError);
```

Return Value

dError	The numeric DWORD value of the last generated function error.

Remarks

Whenever a function error occurs, an error value is generated. You may use RFGetLastError to return the value of the error. The generated errors are as follows:

WLNOERROR	No error returned from function call.
WLCOMMERROR	RF network communication error
WLMEMORYERROR	Memory allocation error
WLNOTINITIALIZED	RF object is improperly initialized.
WLREQUESTFAIL	Input request string format failure
WLINVALIDPARAMS	Invalid parameter(s) for function call
WLINVALIDRETURN	Invalid return designator for function call
WLFUNCTIONFAILED	RF device returned a failure code for the function
WLBCPARSEERROR	Failure parsing barcode file
WLBCADDERROR	Failed to add barcode to object
WLBCCLEARERROR	Failed to clear barcode list
WLBARCODELIMITEXCEEDED	Too many barcodes for configuration file
WLTONEADDERROR	Failed to add tone to object
WLTONECLEARERROR	Failed to clear tone list

WLIOQUEUEERROR	IO queue is empty
WLNOINPUTTYPE	Unable to determine input type
WLPORTTIMEOUT	Auxiliary port timeout
WLDTOUTOFSYNC	Terminal date/time out of sync by more than five seconds

See the Constant Values for the numeric equates returned by the function.

Example

See the RFFile Samples.

RFListFiles Method

This member of RFFile is supported on ALL devices.

The RFListFiles method returns the total number of files saved on an RF device that match the input mask and stores the file names in the current object.

VB

```
nFiles = object.RFListFiles (pszMask)
```

VC++

```
HRESULT hr = object->RFListFiles(LPCTRSTR pszMask, short
*nFiles);
```

Return Value

nFiles	The file count variable. This method will return a -1 if an error occurs. Use the RFGetLastError method to return the generated error code.

Parameters

pszMask	The list file mask (e.g. "*.*" will return all files while "*.ton" will return only tone files). This file mask is similar to the DOS file mask convention.

Remarks

To return the value of the last successful RFListFiles method without making an actual call to the RF device, use the RFFileCount Method. Use the RFListFilesEx Method to additionally store the file date, time and size values within the RFFile object.

Example

See the RFFile Samples.

RFListFilesEx Method

This member of RFFile is supported on ALL devices.

Like the RFListFiles method, the RFListFilesEx method also returns the total number of files saved on an RF device that match the input mask and stores the file names in the current object. In addition, it also stores the date and time the file was last modified and its size. Please see the Remarks for more information on accessing this extended information.

VB

```
nFiles = object.RFListFilesEx (pszMask)
```

VC++

```
HRESULT hr = object->RFListFilesEx(LPCTRSTR pszMask, short
*nFiles);
```

Return Value

nFiles	The file count variable. This method will return a -1 if an error occurs. Use the RFGetLastError method to return the generated error code.

Parameters

pszMask	The list file mask (e.g. "*.*" will return all files while "*.ton" will return only tone files). This file mask is similar to the DOS file mask convention.

Remarks

Use the RFFileDate Method to return the last modification date of a file stored within the RFFile object. Use the RFFileTime Method to return the last modification time of a file stored within the RFFile object. Use the RFFileSize Method to return the size of a file stored within the RFFile object.

RFStoreFile Method

This member of RFFile is supported on ALL devices.

The RFStoreFile method saves the data stored in the pszFileData parameter as a DOS file on an RF device.

VB

```
bStatus = object.RFStoreFile (pszFileName, pszFileData)
```

VC++

```
HRESULT hr = object->RFStoreFile(LPCTSTR pszFileName,
pszFileData, BOOL *bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

pszFileName	The file name that the current RFFile object will be saved as on the RF device. You must specify a 3 character file type extension when passing pszFileName.
pszFileData	The data to be stored.

Remarks

The RFFile object uses the standard DOS 8.3 file naming convention.

Note: Do not use an RFFile object to store the other WaveLink objects (menus, tones, and barcodes). Each of these has its own methods for interacting with the RF device.

Example

```
' VB Sample Code
Dim wlFile As New RFFile
Dim strBuffer As String
.
.
.
  strBuffer = "A string representing data to be stored _
            on an RF device!"
  If wlFile.RFStoreFile ("FLDemo.dat", strBugger) = False Then
    GoTo ExitApp
  End If
```

RFTransferFile Method

This member of RFFile is supported on ALL devices.

The RFTransferFile method allows you to transfer files bi-directionally between a network host and an RF device.

VB

```
bStatus = object.RFTransferFile (pszSourceFile,
pszTargetFile, bToDevice)
```

VC++

```
HRESULT hr = object->RFTransferFile(LPCTSTR pszSourceName,
LPCTSTR pszTargetFile, BOOL bToDevice, BOOL *bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

pszSourceFile	The source file to be transferred. If the source file resides on the host, enter the complete path for the file (e.g. “C:\Temp\File.txt). Alternatively, you may also enter a UNC path. If the source file resides on the local RF device, enter only the file name.
pszTargetFile	The target file of the file transfer. If the target file resides on the host, enter the complete path for the file (e.g. “C:\Temp\File.txt). Alternatively, you may also enter a UNC path. If the target file resides on the local RF device, enter only the file name.
bToDevice	If this value is set to True, the source file will be transferred from the host to the RF client. If this value is set to False, the source file will be transferred from the RF Client to the host.

Example

See the CreateBitmap Method of the WaveLinkFactory object.

RFFile Samples

Using both Visual Basic and Visual C++ sample code, the following set of applications will demonstrate the use and capabilities of the RFFile object and its methods. Both applications will list the current files on an RF device, save a new file to the device, then delete the file from the device.

VB Example

```
' RFFile Object Demo Application
'
' Import a sleep function from the kernel32 dll
Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)
' Declare an RFIO object for use throughout the
' demo application.
Public rfConsole As New RFIO
' Declare an RFFile object for use throughout the
' demo application.
Public rfFil As New rfFile
' strBuffer is a temp storage buffer
Dim strBuffer As String
' nTmp is a temp/ storage buffer for integers
Dim nTmp As Integer

Dim nLcv As Integer
Dim csBuffer As String

Public Sub Main()
' This application demonstrates the use of the various RFFile
' methods. It assumes a display area of 20x8. In practice, one
' should use an RFTerminal object to format output to fit the
' current device.
' The following lines clear the display and output the app title in
'   reverse video.
    rfConsole.RFPrint 0, 0, "   Welcome To The   ", _
              WLCLEAR + WLREVERSE
    rfConsole.RFPrint 0, 1, " RFFile Method Demo ", WLREVERSE
    rfConsole.RFPrint 0, 3, "Hit a key to proceed", WLNORMAL
' Hide the cursor for a cleaner display
    rfConsole.RFPrint 0, 9, "", WLNORMAL
'
' Flush the output buffer and await a keystroke/scan
    If rfConsole.GetEvent() = "" Then
        GoTo ExitApp
    End If  ' If rfConsole.GetEvent() = ""

' The WaveLink RFFile object provides an easy way for applications
```

```
' to store, retrieve, list, and delete files from the RF device.
' Sometimes it is desirable to determine if a particular file
' resides on the RF device. The use of RFListFiles is the
' quickest and easiest method to do this.
    If rfFile.RFListFiles("*.*") < 0 Then ' We're listing all files
        strBuffer = "List Error : " + rfFile.RFGetLastError()
        rfConsole.RFPrint 0, 2, strBuffer, WLCLEAR + WLFLUSHOUTPUT
        Call Sleep(2000)
        GoTo ExitApp
    End If  ' If rfFile.RFListFiles("*.*") < 0
' To find a particular file, pass the file name as the parameter to
' RFListFiles :
'      nFound = rfFile.RFListFiles ("Target.dat")
' Since we just wish to list the files on the current RF device,
' we used the wildcard notation ("*.*").

    nTmp = rfFile.RFFileCount()
' It would have been quicker to use the return value from RFListFile
' instead of an additional function call, but RFFileCount was used
' for demonstration purposes.
    For nLcv = 0 To nTmp
        If (nLcv Mod 7) = 0 Then
            If nLcv Then
                rfConsole.GetEvent
            End If  ' If nLcv
            rfConsole.RFPrint 0, 0, "  RF Device Files:  ", _
                      WLCLEAR + WLREVERSE
        End If   ' If (nLcv Mod 7) = 0
        rfConsole.RFPrint 0, nLcv + 1, rfFile.RFFileName(nLcv), _
                  WLNORMAL
    Next nLcv   ' For nLcv = 0 To nTmp
    rfConsole.GetEvent

' When it is necessary to store data files on the RF device, the
' RFFile object provides methods to store, retrieve,
' and delete them.
    rfConsole.RFPrint 0, 2, "Storing data file...", _
              WLCLEAR + WLFLUSHOUTPUT
    strBuffer = "A string representing data to be stored on an _
                 RF device!"
    If rfFile.RFStoreFile("FLDemo.Dat", csBuffer) = False Then
        strBuffer = "Store Error : " + rfFile.RFGetLastError()
        rfConsole.RFPrint 0, 2, strBuffer, WLCLEAR + WLFLUSHOUTPUT
        Call Sleep(2000)
        GoTo ExitApp
    End If   ' If rfFile.RFStoreFile("FLDemo.Dat", csBuffer) = False
```

```
    rfConsole.RFPrint 0, 3, "Getting data file...", _
              WLNORMAL + WLFLUSHOUTPUT
    strBuffer = rfFile.RFGetFile("FLDemo.Dat")
    If rfFile.RFGetLastError() <> WLNOERROR Then
        strBuffer = "Get Error : " + rfFile.RFGetLastError()
        rfConsole.RFPrint 0, 2, csBuffer, WLCLEAR + WLFLUSHOUTPUT
        Call Sleep(2000)
        GoTo ExitApp
    End If  ' If rfFile.RFGetLastError() <> WLNOERROR

    If strBuffer = "A string representing data to be stored on _
                   an RF device!" Then
        rfConsole.RFPrint 0, 4, "Data Read Success!", WLNORMAL
    Else: rfConsole.RFPrint 0, 4, "Error In Data Read!", WLNORMAL
    End If  ' If strBuffer = "A string representing data to be
            ' stored on an RF device!"

    rfConsole.RFPrint 0, 6, "     Hit a key!     ", WLREVERSE
    rfConsole.RFPrint 0, 9, "", WLNORMAL
    rfConsole.GetEvent

' Anytime an application changes the configuration of an RF device,
'  it should return it to its original values during cleanup
    rfFile.RFDeleteFile "FLDemo.Dat"

' NOTE : Do NOT use an RFFile object to store the other WaveLink
' objects (menus, tones, & barcodes). Each of these has its own
' methods for interacting with the RF device.

ExitApp:
End Sub
```

VC++ Example

```
////////////////////////////////////////////////////////
//   Place RF application's main code here
IRFIO    rfConsole;    // Declare an RFIO interface object
IRFFile  rfFile;       // Declare an RFError interface object
Cstring  csBuffer;     // Storage buffer for data strings
Short    nTmp;         // Temporary short variable

    // Create an RFIO object in the automated server for our use
    if (rfConsole.CreateDispatch ("WAVELINKOLE.RFIO") == FALSE) ↵
        return FALSE;
    // Create an RFFile object in the automated server for our use
    if (rfFile.CreateDispatch ("WAVELINKOLE.RFFILE") == FALSE) {
        // Release the RFIO object
        rfConsole.ReleaseDispatch ();
        return FALSE;
    } // if (rfFile.CreateDispatch ("WAVELINKOLE.RFFILE") == FALSE)

    // This application demonstrates the use of the various RFFile
    // methods. It assumes a display area of 20x8. In practice,
    // one should use an RFTerminal object to format output to fit
    // the current device.
    // The following lines clear the display and output the app
    // title in reverse video.
    rfConsole.RFPrint (0, 0, "   Welcome To The   ", ↵
              WLCLEAR | WLREVERSE);
    rfConsole.RFPrint (0, 1, " RFFile Method Demo ", WLREVERSE);
    rfConsole.RFPrint (0, 3, "Hit a key to proceed", WLNORMAL);
    // Hide the cursor for a cleaner display
    rfConsole.RFPrint (0, 9, "", WLNORMAL);

    // Flush the output buffer and await a keystroke/scan
    if ((rfConsole.GetEvent ()).IsEmpty () == TRUE) {
        // Release the RFIO & RFFile objects
        rfConsole.ReleaseDispatch ();
        rfFile.ReleaseDispatch ();
        return FALSE;
    } // if ((rfConsole.GetEvent ()).IsEmpty () == TRUE)
    // The WaveLink RFFile object provides an easy way for
    // applications to store, retrieve, list, and delete files
    // from the RF device. Sometimes it is desirable to determine
    // if a particular file resides on the RF device. The use of
    // RFListFiles is the quickest and easiest method to do this.
    if (rfFile.RFListFiles ("*.*") < 0) { // We're listing all files
        csBuffer.Format ("List Error : %ld", ↵
                 rfFile.RFRFGetLastError());
```

```
        rfConsole.RFPrint (0, 2, csBuffer, WLCLEAR | WLFLUSHOUTPUT);
        Sleep (2000);
        // Release the RFIO & RFFile objects
        rfConsole.ReleaseDispatch ();
        rfFile.ReleaseDispatch ();
        return FALSE;
    } // if (rfFile.RFListFiles ("*.*") < 0)

    // To find a particular file, pass the file name as the
    // parameter to RFListFiles :     //
bFound = (rfFile.RFListFiles ("Target.dat") == 1);
    // Since we just wish to list the files on the current RF
    // device, we used the wildcard notation ("*.*").

    nTmp = rfFile.RFFileCount ();
    // It would have been quicker to use the return value from
    // RFListFiles instead of an additional function call,
    // but RFFileCount was used for demonstration purposes.
    for (int nLcv = 0; nLcv < nTmp; ++nLcv) {
        if ((nLcv % 7) == 0) {
            if (nLcv) rfConsole.GetEvent ();
            rfConsole.RFPrint (0, 0, "  RF Device Files:  ", ↲
                       WLCLEAR | WLREVERSE);
        } // if ((nLcv % 7) == 0)
        rfConsole.RFPrint (0, nLcv + 1, rfFile.RFFileName (nLcv), ↲
                  WLNORMAL);
    } // for (int nLcv = 0; nLcv < nTmp; ++nLcv)
    rfConsole.GetEvent ();

    // When it is necessary to store data files on the RF device,
    // the RFFile object provides methods to store, retrieve, and
    // delete them.
    rfConsole.RFPrint (0, 2, "Storing data file...", ↲
              WLCLEAR | WLFLUSHOUTPUT);
    csBuffer = "A string representing data to be stored on an ↲
               RF device!";
    if (rfFile.RFStoreFile ("FLDemo.Dat", csBuffer) == FALSE) {
        csBuffer.Format ("Store Error : %ld", ↲
                 rfFile.RFRFGetLastError());
        rfConsole.RFPrint (0, 2, csBuffer, WLCLEAR | WLFLUSHOUTPUT);
        Sleep (2000);
        // Release the RFIO & RFFile objects
        rfConsole.ReleaseDispatch ();
        rfFile.ReleaseDispatch ();
        return FALSE;
    } // if (rfFile.RFStoreFile ("FLDemo.Dat", csBuffer) == FALSE)
```

```
rfConsole.RFPrint (0, 3, "Getting data file...", ↵
          WLNORMAL | WLFLUSHOUTPUT);
if ((csBuffer = rfFile.RFGetFile ("FLDemo.Dat")).IsEmpty ↵
    () == TRUE) {
    csBuffer.Format ("Get Error : %ld", ↵
             rfFile.RFRFGetLastError());
    rfConsole.RFPrint (0, 2, csBuffer, WLCLEAR | WLFLUSHOUTPUT);
    Sleep (2000);
    // Release the RFIO & RFFile objects
    rfConsole.ReleaseDispatch ();
    rfFile.ReleaseDispatch ();
    return FALSE;
} // if ((csBuffer = rfFile.RFGetFile ("FLDemo.Dat")).IsEmpty ↵
         () == TRUE)

if (csBuffer == "A string representing data to be stored on ↵
    an RF device!")
    rfConsole.RFPrint (0, 4, "Data Read Success!", WLNORMAL);
else rfConsole.RFPrint (0, 4, "Error In Data Read!", WLNORMAL);
rfConsole.RFPrint (0, 6, "     Hit a key!     ", WLREVERSE);
rfConsole.RFPrint (0, 9, "", WLNORMAL);
rfConsole.GetEvent ();

// Anytime an application changes the configuration of an
// RF device, it should return it to its original values
// during cleanup
rfFile.RFDeleteFile ("FLDemo.Dat");

// NOTE : Do NOT use an RFFile object to store the other
// WaveLink objects (menus, tones, & barcodes). Each of these
// has its own methods for interacting with the RF device.
```

RFIO Object

The RFIO object is supported on ALL devices.

The RFIO object offers basic input and output methods that will function over a WaveLink wireless network.

Methods

ActivateScanner
AddHotKey
ClearHotKey
GetEvent
EventCount
LastBarcodeType
LastExtendedType
LastInputType
PullScreen
PushScreen
RestoreScreen
RFAux
RFFlushoutput
RFGetLastError
RFInput
RFPrint
RFSpool
SetFillChar
SetInputTimeout
TellEvent

Prog IDs

"WAVELINKOLE.RFIO"

Method Summary

ActivateScanner Method
- Activates the RF device's scanner without the prompt of an RFInput or GetEvent call.

AddHotKey Method
- Defines custom command keys for your applications.

ClearHotKey Method
- Clears custom command keys from the RFIO object.

EventCount Method
- Returns the number of input events in the input queue.

GetEvent Method
- Returns a value after a single key, function key combination, or scan input event has occurred.

LastBarcodeType Method
- Returns the last barcode type scanned by the RF device.

LastExtendedType Method
- Returns the last extended type, either a barcode type or widget ID, of the last input call.

LastInputType Method
- Returns the last input type

PullScreen Method
- Restores a previously saved screen while deleting the screen file from the RF device memory.

PushScreen Method
- Saves the current displayed screen for later restoration.

RestoreScreen Method
- Restores a previously saved screen and does NOT remove the screen file from the RF device memory.

RFAux Method
- Sends data directly to the RF device serial port.

RFFlushoutput Method
- Sends all data in the output buffer to the RF device.

RFGetLastError Method
- Returns the value of the last generated RFIO error.

RFInput Method
- Returns user input from an RF device.

RFPrint Method
- Prints data to the RF device's display.

RFSpool Method
- Spools label data and a copy count to the RF device's serial port.

SetFillChar Method
- Defines the fill character for RFInput input prompts.

SetInputTimeout Method
- Defines the amount of time that elapses before an RFInput input prompt expires.

TellEvent Method
- Checks the input queue for data sent from the RF device without removing the data from the input queue.

ActivateScanner Method

This member of RFIO is supported on ALL devices.

The ActivateScanner method will activate the RF device's scanner for input using the given barcode configuration without the actual prompt of an RFInput or GetEvent call.

VB

```
bStatus = object.ActivateScanner (pszBarCfg)
```

VC++

```
HRESULT hr = object->ActivateScanner(LPCTSTR pszBarCfg, BOOL
*bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

pszBarCfg	The file name of the barcode configuration used by the input call. Enter null (") for this parameter to use the default configuration.

Remarks

This function has been designed primarily for polling scenarios, such as those in which the TellEvent Method is used.

AddHotKey Method

This member of RFIO is supported on ALL devices.

The AddHotKey method lets you custom define additional command keys for your applications.

VB

```
bStatus = object.AddHotKey (nHotKey)
```

VC++

```
HRESULT hr = object->AddHotKey(short HotKey, BOOL *bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

nHotKey	A short integer specifying the ASCII value of the new command hot key (for example, 27)

Remarks

Use the ClearHotKey Method to clear any custom command keys you have created. The return type for custom command keys is WLCOMMANDTYPE. Use the LastInputType Method to process input based on the return type.

Example

```
' VB Sample Code
Dim wlio As New RFIO.
.
.
  If bStatus = wlio.AddHotKey (Asc('Z')) = False Then
    GoTo Exit App
  End If
```

ClearHotKey Method

This member of RFIO is supported on ALL devices.

The ClearHotKey method clears all custom command keys from the current RFIO object.

VB

```
bStatus = object.ClearHotKey ()
```

VC++

```
HRESULT hr = object->ClearHotKey(BOOL *bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Remarks

Use the AddHotKey Method to create custom command keys for your applications.

EventCount Method

This member of WaveLinkIO is supported on ALL devices.

The EventCount method returns the number of input events currently in the input queue.

VB

```
lCount = object.EventCount ()
```

VC++

```
HRESULT hr = object->EventCount(long *lCount);
```

Return Value

nCount	The number of events in the input queue.

Remarks

If multiple input events are in the input queue, subsequent calls to either RFInput or GetEvent within the same WaveLinkIO object are required to return the additional events. Each subsequent call that you make to RFInput or GetEvent will remove one additional event from the input stack.

GetEvent Method

This member of RFIO is supported on ALL devices.

The GetEvent method will immediately return a value after a single key, function key combination, or scan input event has occurred.

VB

```
pszEvent = object.GetEvent ()
```

VC++

```
HRESULT hr = object->GetEvent(Cstring *pszEvent);
```

Return Value

pszEvent	The single key, function key, or scan input value variable. If this function returns an empty string, an error may have occurred. Use the RFGetLastError method to return the generated error code.

Remarks

Unlike a standard input which only returns after the Carriage Return is pressed or the maximum input length has been reached, GetEvent immediately returns a single, function key combination, or a scanner event. GetEvent will return only the first character entered on a keypad input but will return the entire value for a scan input. You may also use the LastInputType Method to return the origin of the last input type (for example, scanner, keypad, function key, etc.)

Example

```
' VB Sample Code
Dim wlio As New RFIO
Dim wlterm As New RFTERMINAL
Dim pszVerifyIn As String
.
.
.
  ' Do something to instruct the user to enter input.
  ' This example uses an RFPrint call.
  wlio.RFPrint 0, (wlterm.TerminalHeight - 1), _
           "Verify y/n ? ",  WLREVERSE + WLCLREOLN
  pszVerifyIn = wlio.GetEvent
```

LastBarcodeType Method

This member of RFIO is supported on ALL devices.

The LastBarcodeType method returns the numeric representation of the last barcode type scanned by the RF device.

VB

```
nType = object.LastBarcodeType ()
```

VC++

```
HRESULT hr = object->LastBarcodeType(short *nType);
```

Return Value

nType	The variable that contains the barcode type (see Remarks). This function returns a -1 if the last input is not scanned input.

Remarks

The possible barcode types are as follows:

UPC_E0
UPC_E1
UPC_A
MSI
EAN_8
EAN_13
CODABAR
CODE_39
CODE_D25
CODE_I25
CODE_11
CODE_93
CODE_128
WLNOSYMBOLOGY

See the Constant Values for the numeric equates returned by the function.

Example

```
' VB Sample Code
Dim wlio As New RFIO
Dim nLastInputType As Integer
Dim pszScanIn As String
pszScanIn = ""
.
.
.
  ' use RFInput to obtain input
  pszScanIn = wlio.RFInput ("", 20, 0, 3, "BCDemo", _
              NORMALKEYS, WLDISABLE_KEY)

  ' do error checking (not shown)
.
.
.
  nLastInputType = wlio.LastInputType
  Select Case nLastInputType
    Case WLCOMMANDTYPE
      If pszScanIn = Chr$(24) Then
        Exit Sub
      End If
    Case WLKEYTYPE

    Case WLSCANTYPE
      If wlio.LastBarcodeType() = CODE_39 Then
        ' do something to process the scan
      End If
      If wlio.LastBarcodeType() = UPC_A Then
        ' do something to process the scan
      End If
  End Select
```

LastExtendedType Method

This member of RFIO is supported on ALL devices.

The LastExtendedType method returns the last extended type of the last input call. The extended type can be a barcode type or a widget ID.

VB

```
nType = object.LastExtendedType ()
```

VC++

```
HRESULT hr = object->LastExtendedType(short *nType);
```

Return Value

nType	The barcode type or widget ID (see Remarks). This function will return a -1 if an error occurs. Use the RFGetLastError method to return the generated error code.

Remarks

When processing input from a widget, use the LastExtendedType method to obtain the widget ID. The widget ID is returned through a previous RFInput call or a GetEvent call.

The possible barcode types are as follows:

UPC_E0

UPC_E1

UPC_A

MSI

EAN_8

EAN_13

CODABAR

CODE_39

CODE_D25

CODE_I25

CODE_11

CODE_93

CODE_128

WLNOSYMBOLOGY

Example

```
' VB Sample Code
Dim wlio As New RFIO
Dim DoneWidget As WaveLinkWidget
Dim StartWidget As WaveLinkWidget
Dim nLastInputType As Integer
Dim pszProcessIn As String
pszProcessIn = ""
.
.
.
  ' create widgets. In this example, create widgets named
  ' StartWidget and DoneWidget (not shown)
.
.
.
  ' use RFInput or GetEvent to obtain input
  pszProcessIn = wlio.GetEvent
  ' do error checking (not shown)
.
.
.
  nLastInputType = wlio.LastInputType
  Select Case nLastInputType
    Case WLCOMMANDTYPE
      If pszProcessIn = Chr$(24) Then
        Exit Sub
      End If
    Case WLKEYTYPE

    Case WLSCANTYPE

    Case WLWIDGETTYPE
      If wlio.LastExtendedType() = StartWidget.WidgetID Then
        GetSignature
      End If
      If wlio.LastExtendedType() = DoneWidget.WidgetID Then
        Exit Sub
      End If
  End Select
```

LastInputType Method

This member of RFIO is supported on ALL devices.

The LastInputType method returns a numeric representation of the last input's specific type.

VB

```
nType = object.LastInputType ()
```

VC++

```
HRESULT hr = object->LastInputType(short *nType);
```

Return Value

nType	The returned input type (see Remarks). This function will return a -1 if an error occurs. Use the RFGetLastError method to return the generated error code.

Remarks

The possible return constants for LastInputType are as follows:

WLCOMMANDTYPE	- Function Key input
WLKEYTYPE	- Standard Key input
WLSCANTYPE	- Scanner input
WLMENUBARTYPE	- Menubar widget input
WLPOPUPTYPE	- Popup trigger widget input
WLWIDGETTYPE	- Widget input
WLTIMEDOUT	- RFInput input prompt expired

See the Constant Values for the numeric equates returned by the function.

Example

```
' VB Sample Code
Dim wlio As New RFIO
Dim nLastInputType As Integer
Dim pszScanIn As String
pszScanIn = ""
.
.
.
  ' use RFInput or GetEvent to obtain input
  pszScanIn = wlio.GetEvent
  ' do error checking (not shown)
.
.
.
  nLastInputType = wlio.LastInputType
  Select Case nLastInputType
    Case WLCOMMANDTYPE
      If pszScanIn = Chr$(24) Then
        Exit Sub
      End If
    Case WLKEYTYPE

    Case WLSCANTYPE
      ProcessScan
  End Select
```

PullScreen Method

This member of RFIO is supported on ALL devices.

The PullScreen method lets you restore a previously saved screen as the current screen while simultaneously deleting the screen file from the RF device's memory.

VB

```
bStatus = object.PullScreen (pszScreen)
```

VC++

```
HRESULT hr = object->PullScreen(LPCTSTR pszScreen, BOOL
*bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

pszScreen	The screen file name to be restored.

Remarks

Use the PushScreen Method to save a current screen file to an RF devices memory. Use the RestoreScreen Method if you wish to restore a screen without deleting it from the RF device's memory. When used in conjunction, PushScreen and PullScreen or RestoreScreen allow you to temporarily interrupt a displayed screen within your application, then restore the screen to its original state. This allows for faster display when re-using screens.

Example

```
' VB Sample Code
Dim wlio As New RFIO
.
.
.
  ' flush the output before using PushScreen.
  ' You can flush the output with:
  ' 1. An input call
  ' 2. The RFFlushoutput method
  ' 3. The RFPrint method when WLFLUSHOUTPUT is set as the
  '    display mode
  wlio.RFPrint 0, 7, "Version 3.01 ", _
       WLNORMAL + WLFLUSHOUTPUT
  wlio.PushScreen "Version"
.
.
.
  ' re-display the Version screen later
  wlio.PullScreen "Version"
```

PushScreen Method

This member of RFIO is supported on ALL devices.

The PushScreen method saves the current displayed screen directly to an RF device's non-volatile memory for later restoration.

VB

```
bStatus = object.PushScreen (pszScreen)
```

VC++

```
HRESULT hr = object->PushScreen(LPCTSTR pszScreen, BOOL
*bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

pszScreen	The file name in which the currently displayed screen will be saved.

Remarks

Use the PullScreen Method to restore a previously saved screen file if you wish to remove the screen file from the RF device's memory. Use the RestoreScreen Method to restore a screen without removing it from the RF device's memory. When used in conjunction, PushScreen and PullScreen or RestoreScreen allow you to temporarily interrupt a displayed screen within your application, then restore the screen to its original state.This allows for faster display when re-using screens.

It is recommended that you clear the output queue immediately before using the PushScreen method.

Example

See the PullScreen Method.

RestoreScreen Method

This member of RFIO is supported on ALL devices.

The RestoreScreen method restores a screen file as the current display screen. RestoreScreen will NOT remove the screen file from the RF device's memory.

VB

```
bStatus = object.RestoreScreen (pszScreen)
```

VC++

```
HRESULT hr = object->RestoreScreen(LPCTSTR pszScreen, BOOL
*bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

pszScreen	The screen file name to be restored.

Remarks

Use the PushScreen Method to save a current screen file to an RF devices memory. Use the PullScreen Method if you wish to restore a screen while deleting it from the RF device's memory. When used in conjunction, PushScreen and PullScreen or RestoreScreen allow you to temporarily interrupt a displayed screen within your application, then restore the screen to its original state. This allows for faster display when re-using screens.

Example

```
' VB Sample Code
Dim wlio As New RFIO
.
.
.
  ' flush the output before using PushScreen.
  ' You can flush the output with:
  ' 1. An input call
  ' 2. The RFFlushoutput method
  ' 3. The RFPrint method when WLFLUSHOUTPUT is set as the
  '    display mode
  wlio.RFPrint 0, 7, "Version 3.01 ", _
       WLNORMAL + WLFLUSHOUTPUT
  wlio.PushScreen "Version"
.
.
.
  ' re-display the Version screen later
  wlio.RestoreScreen "Version"
```

RFAux Method

This member of RFIO is supported on ALL devices.

The RFAux method sends data directly to an RF device's printer port.

VB

```
bStatus = object.RFAux (pszData)
```

VC++

```
HRESULT hr = object->RFAux(LPCTSTR pszData, BOOL *bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

pszData	The string of data to be sent to the RF device's printer port.

Remarks

For devices supporting the RFAux method, the default UART settings are: 9600 BAUD, 8 data bits, 1 stop bit, and no parity. To change the default settings, use the ConfigurePort Method of the RFAuxPort Object.

Note: A series 3000 device with a serial port defaults to 19200 BAUD.

RFFlushoutput Method

This member of RFIO is supported on ALL devices.

The RFFlushoutput method immediately forces output data to be sent to an RF device.

VB

```
bStatus = object.RFFlushoutput ()
```

VC++

```
HRESULT hr = object->RFFlushoutput(BOOL *bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Remarks

For wireless network efficiency purposes, data in an output queue will not be sent to an RF device until either an input call is made or the maximum output size (100 bytes) of a radio packet has been reached. RFFlushoutput allows you to force data in the output queue to be sent without either of these two conditions.

RFGetLastError Method

This member of RFIO is supported on ALL devices.

The RFGetLastError method returns the value of the last generated error.

VB

```
dError = object.RFGetLastError()
```

VC++

```
HRESULT hr = object->GetLastError(DWORD *dError);
```

Return Value

dError	The numeric DWORD value of the last generated function error.

Remarks

Whenever a function error occurs, an error value is generated. You may use RFGetLastError to return the value of the error. The generated errors are as follows:

WLNOERROR	No error returned from function call.
WLCOMMERROR	RF network communication error
WLMEMORYERROR	Memory allocation error
WLNOTINITIALIZED	RF object is improperly initialized.
WLREQUESTFAIL	Input request string format failure
WLINVALIDPARAMS	Invalid parameter(s) for function call
WLINVALIDRETURN	Invalid return designator for function call
WLFUNCTIONFAILED	RF device returned a failure code for the function
WLBCPARSEERROR	Failure parsing barcode file
WLBCADDERROR	Failed to add barcode to object
WLBCCLEARERROR	Failed to clear barcode list
WLBARCODELIMITEXCEEDED	Too many barcodes for configuration file
WLTONEADDERROR	Failed to add tone to object
WLTONECLEARERROR	Failed to clear tone list

WLIOQUEUEERROR	IO queue is empty
WLNOINPUTTYPE	Unable to determine input type
WLPORTTIMEOUT	Auxiliary port timeout
WLDTOUTOFSYNC	Terminal date/time out of sync by more than five seconds

See the Constant Values for the numeric equates returned by the function.

Example

```
' VB Sample Code
wlio As New RFIO
nError As Integer
.
.
.
  ' in this example, we use an RFInput call
  pszWelcomeIn = wlio.RFInput(pszWelcomeInDefault, 1, 1, _
                 33, "BarCode", WLCAPSLOCK, WLNO_RETURN_BKSP)
  ' it is critical to check for errors after communication
  ' occurs between the app and the RF device.
  nError = wlio.RFGetLastError
  If nError <> WLNOERROR Then
    GoTo ExitApp
  Else
  ' process input
```

RFInput Method

This member of RFIO is supported on ALL devices.

The RFInput method returns user input from an RF device used over a wireless network.

VB

```
pszInput = object.RFInput (pszDefault, nLength, nColumn,
nRow, pszBarCfg, nKeyMode, nInputMode)
```

VC++

```
HRESULT hr = object->RFInput(LPCTSTR pszDefault, short
nLength, short nColumn, short nRow, LPCTSTR pszBarCfg, short
nKeyMode, short nInputMode, Cstring *pszInput);
```

Return Value

pszInput	The returned input value variable. If this method returns an empty string an error may have occurred. Use the RFGetLastError method to return the generated error code.

Parameters

pszDefault	Default values to appear in the input prompt (“"). The default value will not be included in the returned input.
nLength	The maximum number of characters that may be entered at the input point by the user before input is automatically returned. This parameter also defines the number of characters that will be displayed at the input point.
nColumn	The on-screen column coordinate in which the input prompt will begin. Columns are numbered from the left starting with column 0.
nRow	The on-screen row coordinate in which the input prompt will begin. Rows are numbered from the top starting with row 0.
pszBarCfg	The file name of the barcode configuration used by the input call. Enter null (“) for this parameter to use the current barcode configuration.
nKeyMode	The current keyboard shift state as represented by a constant (WLNORMALKEYS or WLCAPSLOCK).

nInputMode	The current state for input device options as represented by constants (see Remarks). Multiple options may be used by ORing the desired values. For example, to disable the scanner and function keys enter "WLDISABLE_SCAN+WLDISABLE_FKEYS".

Remarks

The possible values for nKeyMode are:

WLNORMALKEYS	Caps lock off
WLCAPSLOCK	Caps lock on

The possible values for nInputMode are:

WLDISABLE_SCAN	Disable the RF device scanner
WLDISABLE_KEY	Disable the RF device keypad
WLSUPRESS_ECHO	Disable display echo of input
WLNO_RETURN_FILL	Input will not be returned until the enter key is pressed, even if the maximum input length has been reached.
WLNO_RETURN_BKSP	Input will not be returned if the Backspace key is pressed in the first position of the input prompt.
WLDISABLE_FKEYS	Disable function keys
WLFORCE_ENTRY	Input will not return unless an actual value has been entered in the input prompt.
WLALPHA_ONLY	Accept only alphabetic input
WLNUMERIC_ONLY	Accept only numeric input
WLBACKLIGHT	Enable the device's display backlight
WLMAXLENGTH	RFInput will not accept more characters than defined in the nLength parameter.
WLINCLUDE_DATA	RFInput will return both data entered at an input prompt and any additional function key input.
WLSOFT_TRIGGER	This input mode will automatically activate the RF device's scanner when the RFInput function is called.
WLCLR_INPUT_BUFFER	Clears the input buffer of any pending data.
WLINGORE_CRLF	Includes any carriage returns or line feeds embedded within the returned input string. By

	default, returned input strings containing these characters are broken into separate input packets.
WLECHO_ASTERISK	Echo an asterisk (“*) to RF device screen with each key entered.
WLNO_NONPRINTABLE	Suppress non-printable characters. This input mode is used primarily when using foreign character sets.

RFInput will return input immediately after either the Enter key is pressed, the maximum input length has been reached, the Backspace key is pressed in the first position, or a function key is pressed. If a function key is pressed, the RFInput will only return the value for the function key pressed, discarding any data entered in the input prompt (use the WLINCLUDE_DATA to return both the data entered in the input prompt and the function key value). It is important to note that if you use WLINCLUDE_DATA, only the data entered in the input prompt will be removed from the input stack on the initial call to RFInput. A subsequent call to either RFInput or GetEvent within the same RFIO object will be required to remove the function key from the input stack.

You may limit barcode input types through the use of Barcode Configuration files, please see the RFBarcode Object for more information.

To return a single key, function key combination, or scan input use the GetEvent Method.

Use the TellEvent Method to check for user input without removing the input from the input queue and pausing your application.

Example

```
' VB Sample Code
Dim wlio As New RFIO
Dim wlTerm As New RFTerminal
Dim PszVerifyInDefault As String
Dim PszVerifyIn As String
pszVerifyInDefault = ""
.
.
.
  ' Do something to direct the user to enter input. This
  ' example uses a call to RFPrint.
  ' Note: This example uses the RFTerminal object to dynamically
  ' position the screen output.
  wlio.RFPrint 0, (wlTerm.TerminalHeight - 1), _
           "Verify y/n ? ",  WLREVERSE + WLCLREOLN
  ' position the input field off screen
  pszVerifyIn = wlio.RFInput(pszVerifyInDefault, 1, 0, _
                wlTerm.TerminalHeight + 2, "", _
                WLDISABLESCAN + WLNO_RETURN_BKSP)
```

RFPrint Method

This member of RFIO is supported on ALL devices.

The RFPrint method allows you to print data directly to an RF device's display over a wireless network.

VB

```
bStatus = object.RFPrint (nColumn, nRow, pszMessage, nMode)
```

VC++

```
HRESULT hr = object->RFPrint(short nColumn, short nRow,
LPCTSTR pszMessage, short nMode, BOOL *bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

nColumn	The on-screen column coordinate at which output will begin. Columns are numbered from the left starting with column 0.
nRow	The on-screen row coordinate at which output will begin. Rows are numbered from the top starting with row 0.
pszMessage	The displayed message text.
nMode	The display mode of the printed text (see Remarks). Multiple options may be used by ORing the desired constants. For example, to clear the screen then print text in reverse enter "WLCLEAR+WLNORMAL".

Remarks

The possible values for nMode are:

WLCLEAR	Clears the RF device's display
WLNORMAL	Normal display of text
WLREVERSE	Text is displayed in reverse colors
WLCLREOLN	Clear text to the end of current line.

WLFLUSHOUTPUT Automatically flushes the output buffer

Example

```
' VB Sample Code
Dim wlio As New RFIO
Dim wlterm As New RFTerminal
.
.
.
  ' Note: This example uses the RFTerminal object to dynamically
  ' position the screen output.
  wlio.RFPrint 0, 0, "WaveLink Corporation", WLCLEAR + WLREVERSE
  wlio.RFPrint 0, 1, "   Auto Detailing   ", WLNORMAL
  wlio.RFPrint 0, (wlterm.TerminalHeight -1), _
       "   Press any key   ", WLNORMAL

  ' Flush the output buffer and await a keystroke/scan
  If wlio.GetEvent() = "" Then
    GoTo ExitApp
  End If
```

RFSpool Method

This member of RFIO is supported on ALL devices.

The RFSpool method lets you spool label data and a copy count directly to a RF device's printer port. The format of each label is specified using a previously stored printer format file.

VB

```
nStatus = object.RFSpool (pszFileName, nCopies, pszData)
```

VC++

```
HRESULT hr = object->RFSpool(LPCTSTR pszFileName, short
nCopies, LPCTSTR pszData, short *nStatus);
```

Return Value

nStatus	The returned status of the function represented as a numeric value. The possible return values are as follows: 0 - Function executed successfully. -1 - A communication or other WaveLink Server side error occurred. 2 - An error occurred at the client side.

Parameters

pszFileName	The file name of the local printer format file.
nCopies	A short integer representing the number of labels to be spooled to the printer.
pszData	The data for each label to be formatted using the printer format files specified by pszFileName.

Remarks

Printer format files are printer-specific. See your printer documentation for more information.

SetFillChar Method

This member of RFIO is supported on ALL devices.

The SetFillChar method defines the fill character for RFInput input prompts.

VB

```
bStatus = object.SetFillChar (pszFillChar)
```

VC++

```
HRESULT hr = object->SetFillChar(LPCTSTR pszFillChar, BOOL
*bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

pszFillChar	The fill character, passed in quotations.

Example

See RFIO Samples.

SetInputTimeout Method

This member of RFIO is supported on ALL devices.

The SetInputTimeout method defines the amount of time, in seconds, that elapse before an RFInput input prompt expires.

VB

```
bStatus = object.SetInputTimeout (lTimeout)
```

VC++

```
HRESULT hr = object->SetInputTimeout(long lTimeout, BOOL
*bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

lTimeout	The input timeout, in seconds.

Remarks

By default, RFInput prompts do not expire.

An expired RFInput prompt will return an input type of WLTIMEDOUT. Use the LastInputType method to check for the returned input type.

Note: The Symbol 8100 does not currently support this method.

TellEvent Method

This member of RFIO is supported on ALL devices.

The TellEvent method checks the input queue for any data sent from the RF device. It returns immediately whether data was present or not. It does NOT remove the event from the input queue.

VB

```
pszEvent = object.TellEvent ()
```

VC++

```
HRESULT hr = object->TellEvent(Cstring *pszEvent);
```

Return Value

pszEvent	The return input event. If this method returns an empty string an error may have occurred. Use the RFGetLastError method to return the generated error code.

Remarks

As mentioned in the description, TellEvent does not remove any returned input values from the input queue. This method therefore allows you to simply check for user input without pausing your application. To actually remove returned input from the input queue, a subsequent call to another input method, such as GetEvent or RFInput, must be made following TellEvent. You may also use a call to the LastInputType method to return the origin of the last input type (e.g. scan, keypad, function key input).

Example

See RFError Samples.

RFIO Samples

Using both Visual Basic and Visual C++ sample code , the following set of applications will demonstrate the implementation and capabilities of the RFIO object and its methods.

VB Example

```
' RFIO Object Demo Application
'
' Import a sleep function from the kernel32 dll
Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)
' Declare an RFIO object for use throughout the
' demo application.
Public rfConsole As New RFIO
' strBuffer is a temp storage buffer
Dim strBuffer As String


Public Sub Main()
' This application demonstrates the use of the various RFIO methods.
' It assumes a display area of 20x8. In practice, one should use
' an RFTerminal object to format the output to fit the current
' RF device.
'
' The following lines clear the display and output the
' application's title in reverse video.
    rfConsole.RFPrint 0, 0, "Welcome To The RFIO ", _
              WLCLEAR + WLREVERSE
    rfConsole.RFPrint 0, 1, "    Method Demo     ", WLREVERSE
    rfConsole.RFPrint 0, 3, "Hit a key to proceed", WLNORMAL
    ' Hide the cursor for a cleaner display
    rfConsole.RFPrint 0, 9, "", WLNORMAL
'
' Flush the output buffer and await a keystroke/scan
    If rfConsole.GetEvent() = "" Then
    ' Exit the application
        GoTo ExitApp
    End If  ' If rfConsole.GetEvent() = Null Then
'
' There are 4 ways data is "flushed" to the RF device's screen:
'   1. The number of bytes in the output buffer reaches 100;
'   2. The output mode includes the WLFLUSHOUTPUT constant;
'   3. A call to the RFFlushoutput method is made;
'   4. Or, input is requested using the RFInput or GetEvent methods.
'
```

```
' This demonstrates method 2 (described above)
    rfConsole.RFPrint 0, 0, "Enter Your Name :", _
              WLCLEAR + WLFLUSHOUTPUT
    rfConsole.SetFillChar "%"   ' By default the value is "_"
    strBuffer = rfConsole.RFInput("Your Name", 14, 3, 2, "",
                NORMALKEYS, WLBACKLIGHT + WLMAXLENGTH + _
                WLNO_RETURN_BKSP + WLALPHA_ONLY)
                ' "Your Name"    - Default value for input prompt
                ' 14             - Input field length
                ' 3              - Input field start column
                ' 2              - Input field start row (numbered
                '                  from the top of the display)
                ' Null           - The barcode configuration
                '                  name (default in this case)
                ' NORMALKEYS     - The keyboard state
                ' WLBACKLIGHT +  - Input mode : Backlight on +
                ' WLMAXLENGTH +  - Allow a max of 14 characters +
                ' WLNO_RETURN_BKSP +  - Do not return if backspace
                '                       is keyed in position 1 +
                ' WLALPHA_ONLY        - Since names don't generally
                '                       include numbers, accept
                '                       letters only
' Check for function or network errors
    If strBuffer = "" Then
    ' Exit the application
        GoTo ExitApp
    Else
        rfConsole.RFPrint 0, 4, "Hello " + strBuffer, WLNORMAL
        rfConsole.RFPrint 0, 9, "", WLNORMAL + WLFLUSHOUTPUT
        Call Sleep(2000)
    End If  ' If strBuffer = Null Then
'
' It is often desirable to send data to a printer attached to
' the RF device
    rfConsole.RFPrint 0, 0, "Scan A Barcode:", WLCLEAR
    rfConsole.SetFillChar "#"
    strBuffer = rfConsole.RFInput("", 20, 0, 2, "", NORMALKEYS, _
                WLBACKLIGHT + WLDISABLE_KEY + WLFORCE_ENTRY)
    If strBuffer = "" Then
    ' Exit the application
        GoTo ExitApp
    Else
        rfConsole.RFPrint 0, 4, "Data Entered :", WLNORMAL
        rfConsole.RFPrint 0, 5, strBuffer, WLNORMAL
        rfConsole.RFPrint 0, 9, "", WLFLUSHOUTPUT
    '
```

```
    ' Now send the data to printer
        If rfConsole.RFAux(strBuffer) = False Then
            rfConsole.RFPrint 0, 1, "Printer Error: " +
                      rfConsole.RFGetLastError(), WLNORMAL
            rfConsole.RFPrint 0, 3, "Hit a key to proceed", WLNORMAL
            rfConsole.RFPrint 0, 9, "", WLNORMAL
            rfConsole.GetEvent
        End If  'If rfConsole.RFAux(strBuffer) = False Then
    '
        Call Sleep(2000)
    End If  'If strBuffer = Null Then
'
' Explicitly repainting the same screen multiple times is
' excessively taxing on the RF network. A faster and more efficient
' technique is to use the PushScreen, RestoreScreen & PullScreen
' methods.
' First, display the desired screen using RFPrint ...
    rfConsole.RFPrint 0, 0, "This is line one", WLCLEAR
    rfConsole.RFPrint 0, 2, "The quick brown fox ", WLREVERSE
    rfConsole.RFPrint 0, 3, "jumped over the lazy", WLNORMAL
    rfConsole.RFPrint 0, 4, "        dog!        ", WLREVERSE
    rfConsole.RFPrint 0, 7, "Hit a key to proceed", WLNORMAL
    rfConsole.RFPrint 0, 9, "", WLNORMAL
    rfConsole.GetEvent
'
' Second, store the screen on the RF device ...
    rfConsole.PushScreen "RFIODemo"
'
' Lastly, restore it at will.
    rfConsole.RFPrint 0, 3, "Hit a key to proceed", WLCLEAR
    rfConsole.RFPrint 0, 9, "", WLNORMAL
    rfConsole.GetEvent

    rfConsole.PullScreen "RFIODemo"
    rfConsole.GetEvent
'
' NOTES:
'   1. It is not necessary to use GetEvent to flush the output;
'      PullScreen displays the screen automatically. We are just
'      allowing the user to view the restored screen before exiting.
'   2. The screen file is deleted when the PullScreen method
'      is used. If the screen must be restored repeatedly, use
'      RestoreScreen (which does NOT delete it) until
'      it is no longer needed.

ExitApp:
```

```
End Sub
```

VC++ Example

```
////////////////////////////////////////////////////////
//        Place RF application's main code here
IRFIO          rfConsole;        // Declare an RFIO interface object
Cstring        csBuffer;         // Storage buffer for data strings

    // Create an RFIO object in the automated server for our use
    if (rfConsole.CreateDispatch ("WAVELINKOLE.RFIO") == FALSE) ↵
        return FALSE;

    // This application demonstrates the use of the various RFIO
    // methods. It assumes a display area of 20x8. In practice,
    // one should use an RFTerminal object to format output to fit
    // the current device.
    // The following lines clear the display and output the
    // app title in reverse video.
    rfConsole.RFPrint (0, 0, "Welcome To The RFIO ", ↵
              WLCLEAR | WLREVERSE);
    rfConsole.RFPrint (0, 1, "  Method Demo App   ", WLREVERSE);
    rfConsole.RFPrint (0, 3, "Hit a key to proceed", WLNORMAL);
    // Hide the cursor for a cleaner display
    rfConsole.RFPrint (0, 9, "", WLNORMAL);

    // Flush the output buffer and await a keystroke/scan
    if ((rfConsole.GetEvent ()).IsEmpty () == TRUE) {
        // Release the RFIO object
        rfConsole.ReleaseDispatch ();
        return FALSE;
    } // if ((rfConsole.GetEvent ()).IsEmpty () == TRUE)
    // There are 4 ways data is "flushed" to the RF device's screen:
    //   1. The number of bytes in the output buffer reaches 100;
    //   2. The output mode includes the WLFLUSHOUTPUT constant;
    //   3. A call to the RFFlushoutput method is made;
    //   4. Or, input is requested using the RFInput or GetEvent
    //      methods.

    // This demonstrates method 2 (described above)
    rfConsole.RFPrint (0, 0, "  Enter Your Name : ", ↵
              WLCLEAR |  WLFLUSHOUTPUT);
    rfConsole.SetFillChar ("%"); // By default, the fill character
                                 // is "_"
    csBuffer = rfConsole.RFInput ("Your Name",  // Default value
```

```
                                                // for input prompt
            14,     // Input field length
            3,      // Input field start column
            2,      // Input field start row (numbered from the
                    // top of the display)
            NULL,   // The barcode configuration file name
                    // (default in this case)
            NORMALKEYS,         // The keyboard state
            WLBACKLIGHT |       // Input mode : Backlight on |
            WLMAXLENGTH |       // Allow a max of 14 characters |
            WLNO_RETURN_BKSP |  // Do not return if Backspace is
                                // keyed in position 1 |
            WLALPHA_ONLY);      // Since names don't generally
                                // include numbers, accept
                                // letters only
// Check for function or network errors
if (csBuffer.IsEmpty () == TRUE) {
    // Release the RFIO object
    rfConsole.ReleaseDispatch ();
    return FALSE;
} // if (csBuffer.IsEmpty () == TRUE)
else {
    rfConsole.RFPrint (0, 4, CString ("Hello ") + csBuffer, ↵
              WLFLUSHOUTPUT);
    rfConsole.RFPrint (0, 9, "", WLNORMAL);
    Sleep (2000);  // Allow user to read response
} // else

// It is often desirable to send data to a printer attached to
// the RF device
rfConsole.RFPrint(0, 0, "   Scan A Barcode : ", WLCLEAR);
rfConsole.SetFillChar("#"); // By default, the fill character
                            // is "_"
csBuffer = rfConsole.RFInput (NULL, // Default value for input
                                    // prompt
            20,     // Input field length
            0,      // Input field start column
            2,      // Input field start row (numbered from
                    // the top of the display)
            NULL,          // The barcode configuration file
                           // name (default in this case)
            NORMALKEYS,        // The keyboard state
            WLBACKLIGHT |      // Input mode : Backlight on |
            WLDISABLE_KEY |    // Disable normal keyboard input |
            WLFORCE_ENTRY);     // Ensure data will be entered
// Check for function or network errors
```

```
if (csBuffer.IsEmpty () == TRUE) {
    // Release the RFIO object
    rfConsole.ReleaseDispatch ();
    return FALSE;
} // if (csBuffer.IsEmpty () == TRUE)
else {
    rfConsole.RFPrint (0, 4, "Data Entered : ", WLNORMAL);
    rfConsole.RFPrint (0, 5, csBuffer, WLNORMAL);
    rfConsole.RFPrint (0, 9, "", WLFLUSHOUTPUT);
    // Now we shall send it to the printer ...
    if (rfConsole.RFAux (csBuffer) == FALSE) {
        // Display the RFIO error code for RFAux failure
        csBuffer.Format ("Printer Error : %ld", ↵
                 rfConsole.RFRFGetLastError());
        rfConsole.RFPrint (0, 1, csBuffer, WLNORMAL);
        rfConsole.RFPrint (0, 3, "Hit a key to proceed", ↵
                  WLNORMAL);
        rfConsole.RFPrint (0, 9, "", WLNORMAL);
        rfConsole.GetEvent ();         // Wait for single event
    } // if (rfConsole.RFAux (csBuffer) == FALSE)
    Sleep (2000);  // Allow user to read response
} // else

// Explicitly repainting the same screen multiple times
// is excessively taxing on the RF network. A faster and more
// efficient technique is to use the RFPushScreen,
// RF RestoreScreen & RFPullScreen methods.
// First, display the desired screen using RFPrint ...
rfConsole.RFPrint(0, 0, "  This is line one  ", WLCLEAR);
rfConsole.RFPrint(0, 2, "The quick brown fox ", WLREVERSE);
rfConsole.RFPrint(0, 3, "jumped over the lazy", WLNORMAL);
rfConsole.RFPrint(0, 4, "       dog!         ", WLREVERSE);
rfConsole.RFPrint(0, 7, "Hit a key to proceed", WLNORMAL);
rfConsole.RFPrint(0, 9, "", WLNORMAL);
rfConsole.GetEvent();       // Wait for single event

// Second, store the screen on the RF device ...
rfConsole.PushScreen ("RFIODemo");

// Lastly, restore it at will.
rfConsole.RFPrint (0, 3, "Hit a key to proceed", WLCLEAR);
rfConsole.RFPrint (0, 9, "", WLNORMAL);
rfConsole.GetEvent ();      // Wait for single event

rfConsole.PullScreen ("RFIODemo");
rfConsole.GetEvent ();      // Wait for single event
```

```
// NOTES:
//  1. It is not necessary to use GetEvent to flush the output;
//     PullScreen displays the screen automatically. We are just
//     allowing the user to view the restored screen before exiting
//
//  2. The screen file is deleted when the PullScreen method
//     is used. If the screen must be restored repeatedly, use
//     RestoreScreen (which does NOT delete it) until
//     it is no longer needed.

    // Release the RFIO object
    rfConsole.ReleaseDispatch ();
```

RFMenu Object

The RFMenu object is supported on ALL devices.

The RFMenu object lets you easily create and use menus on an RF device over a WaveLink wireless network. RFMenu objects may also be saved directly into the memory of network RF devices for true client-server efficiency within your applications.

Methods

AddOption
AddTitleLine
ClearOptions
ClearTitle
DeleteMenu
DoMenu
GetMenuHeight
GetMenuOption
GetMenuWidth
GetStartColumn
GetStartRow
ListMenuFiles
MenuFileCount
MenuFileName
ResetMenu
RFGetLastError
SetCoordinates
SetMenuHeight
SetMenuWidth
SetStartColumn
SetStartRow
StoreMenu

Prog IDs

"WAVELINKOLE.RFMENU"

Remarks

An RFMenu object is comprised of titles and options. A title is a line or lines of text displayed at the top of a menu. An RFMenu object may contain more than one title line. Options are the available choices or selections that may be returned by the menu. When selected, an option will return the numeric value of its option line. For example, selecting the second option in a three option menu will return the value two to your application.

Method Summary

AddOption Method
- Adds a menu option to the RFMenu object.

AddTitleLine Method
- Adds a single title line to an RFMenu object.

ClearOptions Method
- Removes all menu options from the RFMenu object.

ClearTitle Method
- Clears all title lines from the RFMenu object.

DeleteMenu Method
- Removes menu files from an RF device.

DoMenu Method
- Executes a menu file from an RF device's non-volatile memory and returns the selected option to your application.

GetMenuHeight Method
- Returns the height of the RFMenu object.

GetMenuOption Method
- Returns the name of the selected menu option.

GetMenuWidth Method
- Returns the width of the RFMenu object.

GetStartColumn Method
- Returns the starting column of the RFMenu object.

GetStartRow Method
- Returns the starting row of the RFMenu object.

ListMenuFiles Method
- Returns the total number of menu files on the device and stores the list of names in the RFMenu object.

MenuFileCount Method
- Returns the total number of menu files returned by the last call to the ListMenuFiles method.

MenuFileName Method
- Returns the name of a specific menu file.

ResetMenu Method
- Resets the current RFMenu object.

RFGetLastError Method
- Returns the value of the last generated RFMenu error.

SetCoordinates Method
- Sets the position, height, and width of the RFMenu object.

SetMenuHeight Method
- Defines the height of the RFMenu object.

SetMenuWidth Method
- Defines the width of the RFMenu object.

SetStartColumn Method
- Defines the starting column coordinate of the RFMenu object.

SetStartRow Method
- Defines the starting row coordinate of the RFMenu object.

StoreMenu Method
- Saves the current RFMenu object as a menu file on the RF device for later use.

AddOption Method

This member of RFMENU is supported on ALL devices.

The AddOption method adds additional options to an RFMenu object.

VB

```
bStatus = object.AddOption (pszOption)
```

VC++

```
HRESULT hr = object->AddOption(LPCTSTR pszOption, BOOL
*bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

pszOption	The option name

Remarks

Options are the available choices or selections that may be returned by the menu. Each RFMenu option will return the numeric value of its option line. For example, if you had a menu with a total of three options and the user selected the second option in the menu, the number 2 would be returned to your application.

Example

See the DoMenu Method.

AddTitleLine Method

This member of RFMenu is supported on ALL devices.

The AddTitleLine method adds a single title line to an RFMenu object.

VB

```
bStatus = object.AddTitleLine (pszTitleLine)
```

VC++

```
HRESULT hr = object->AddTitleLine(LPCTSTR pszTitleLine, BOOL
*bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

pszTitleLine	Menu title line text.

Remarks

Titles will always be printed at the top of a menu. For multi-lined titles an RFMenu object may contain additional title lines.

Example

See the DoMenu Method.

ClearOptions Method

This member of RFMenu is supported on ALL devices.

The ClearOptions method removes all options from the RFMenu object.

VB

```
bStatus = object.ClearOptions ()
```

VC++

```
HRESULT hr = object->ClearOptions(BOOL *bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Remarks

As noted in the description, ClearOptions will remove **ALL** options from an RFMenu object.

ClearTitle Method

This member of RFMenu is supported on ALL devices.

The ClearTitle method clears all title lines from the RFMenu object.

VB

```
bStatus = object.ClearTitle ()
```

VC++

```
HRESULT hr = object->ClearTitle(BOOL *bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Remarks

As noted in the description, ClearTitle will remove **ALL** titles from an RFMenu object.

DeleteMenu Method

This member of RFMenu is supported on ALL devices.

The DeleteMenu method removes menu files from an RF device.

VB

```
bStatus = object.DeleteMenu (pszMenuName)
```

VC++

```
HRESULT hr = object->DeleteMenu(LPCTSTR pszMenuName, BOOL
*bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

pszMenuName	The menu file to be deleted. You may pass a single eight character file name or enter a wildcard symbol ("*") to delete all menu files from an RF device.

Example

See the RFMenu Samples.

DoMenu Method

This member of RFMenu is supported on ALL devices.

The DoMenu method executes a menu file from an RF device's non-volatile memory and will return a selected option to your application.

VB

```
nOption = object.DoMenu (pszMenuName)
```

VC++

```
HRESULT hr = object->DoMenu(LPCTSTR pszMenuName, short
*nOption);
```

Return Value

nOption	The numeric value of the option selected. This method will return a -1 if “CTRL-X or the “CLR key is hit. This method will return a -2 if an error occurs. Use the RFGetLastError method to return the error code generated.

Parameters

pszMenuName	The executed menu file. This file name may be up to 8 characters in length.

Remarks

The DoMenu method will return to your application the numeric value of the option line selected by the user. For example, if you had a menu with a total of three options and the user selected the second option in the menu, the number 2 would be returned to your application.

Example

```
' VB Sample Code
Dim rfMnu As New RFMENU
Dim nTmp As Integer
.
.
  rfMnu.ResetMenu
  rfMnu.AddOption "Option 1"
  rfMnu.AddOption "Option 2"
  rfMnu.AddOption "Option 3"
  rfMnu.AddOption "Option 4"
  rfMnu.AddTitleLine "Demonstration"
  rfMnu.SetCoordinates (0, 1, 20, 6);
  ' The menu must be stored on the RF device before it
  ' may be used.
  If rfMnu.StoreMenu("MenuDemo") = False Then
    GoTo ExitApp
  End If
  nTmp = rfMnu.DoMenu("MenuDemo")
```

GetMenuHeight Method

This member of RFMenu is supported on ALL devices.

The GetMenuHeight method returns the height of the current RFMenu object.

VB

```
nHeight = object.GetMenuHeight ()
```

VC++

```
HRESULT hr = object->GetMenuHeight (short *nHeight);
```

Return Value

nHeight	The menu height variable. This method will return a -1 if an error occurs. Use the RFGetLastError method to return the generated error code.

GetMenuOption Method

This member of RFMenu is supported on ALL devices.

The GetMenuOption method returns the name of the specified menu option.

VB

```
pszName = object.GetMenuOption (nOption)
```

VC++

```
HRESULT hr = object->GetMenuOption (short nOption, Cstring
*pszName);
```

Return Value

pszName	The option name. This method will return an empty string if an error occurs. Use the RFGetLastError method to return the generated error code.

Parameters

nOption	The numeric value of the option selected.

Remarks

Options are the available choices or selections that may be returned by the menu. The first menu option is option 1, the second is option 2, etc.

GetMenuWidth Method

This member of RFMenu is supported on ALL devices.

The GetMenuWidth method returns the width, in characters, of the current RFMenu object.

VB

```
nWidth = object.GetMenuWidth ()
```

VC++

```
HRESULT hr = object->GetMenuWidth (short *nWidth);
```

Return Value

nWidth	The returned menu width variable. This method will return a -1 if an error occurs. Use the RFGetLastError method to return the generated error code.

GetStartColumn Method

This member of RFMenu is supported on ALL devices.

The GetStartColumn method returns the starting column for the current RFMenu object.

VB

```
nColumn = object.GetStartColumn ()
```

VC++

```
HRESULT hr = object->GetStartColumn(short *nColumn);
```

Return Value

nColumn	The variable name of the starting column coordinate. This method will return a -1 if an error occurs. Use the RFGetLastError method to return the generated error code.

GetStartRow Method

This member of RFMenu is supported on ALL devices.

The GetStartRow method will return the starting row for the current RFMenu object.

VB

```
nRow = object.GetStartRow ()
```

VC++

```
HRESULT hr = object->GetStartRow(short *nRow);
```

Return Value

nRow	The variable name of the starting row coordinate. This method will return a -1 if an error occurs. Use the RFGetLastError method to return the generated error code.

ListMenuFiles Method

This member of RFMenu is supported on ALL devices.

The ListMenuFiles method returns the total number of menu files saved on an RF device and stores a list of the file names in the current object.

VB

```
nMenus = object.ListMenuFiles ()
```

VC++

```
HRESULT hr = object->ListMenuFiles(short *nMenus);
```

Return Value

nMenus	The variable for the number of menu files stored on an RF device. This method will return a -1 if an error occurs. Use the RFGetLastError method to return the generated error code.

Remarks

To return the value of the last successful ListMenuFiles method without making an actual call to the RF device, use the MenuFileCount Method.

MenuFileCount Method

This member of RFMenu is supported on ALL devices.

The MenuFileCount method returns the total number of menu files successfully returned by the last call to the ListMenuFiles method.

VB

```
nMenuFiles = object.MenuFileCount ()
```

VC++

```
HRESULT hr = object->MenuFileCount(short *nMenuFiles);
```

Return Value

nMenuFiles	The menu file count variable.

Remarks

The MenuFileCount method will return the file count from the last successful call to th ListMenuFiles Method. That is, MenuFileCount will return the last known number of files stored on an RF device without actually communicating via RF to the device. Use the ListMenuFiles method to return the total number of menu files that are currently stored on an RF device.

MenuFileName Method

This member of RFMenu is supported on ALL devices.

The MenuFileName method returns the name for a specific menu file.

VB

```
pszName = object.MenuFileName (nMenu)
```

VC++

```
HRESULT hr = object->MenuFileName(short nMenu, Cstring
*pszName);
```

Return Value

pszName	The menu file name variable. If this method returns an empty string an error may have occurred. Use the RFGetLastError method to return the generated error code.

Parameters

nMenu	The zero based index value of the menu file name being returned. For example, to return the third menu file name, you would pass a 2.

Remarks

For a complete listing of all menu files on an RF device simply loop the MenuFileName method for the entire number of files returned by MenuFileCount.

ResetMenu Method

This member of RFMenu is supported on ALL devices.

The ResetMenu method resets all properties for the current RFMenu object.

VB

```
bStatus = object.ResetMenu ()
```

VC++

```
HRESULT hr = object->ResetMenu(BOOL *bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Remarks

When called, ResetMenu will clear all menu options and titles and reset all menu coordinates to zero. This allows you to re-use a menu object instead of creating a new one.

Example

See the DoMenu Method.

RFGetLastError Method

This member of RFMenu is supported on ALL devices.

The RFGetLastError method returns the value of the last generated error.

VB

```
dError = object.RFGetLastError()
```

VC++

```
HRESULT hr = object->GetLastError(DWORD *dError);
```

Return Value

dError	The numeric DWORD value of the last generated function error.

Remarks

When ever a function error occurs, an error value is generated. You may use RFGetLastError to return the value of the error. The generated errors are as follows:

WLNOERROR	No error returned from function call.
WLCOMMERROR	RF network communication error
WLMEMORYERROR	Memory allocation error
WLNOTINITIALIZED	RF object is improperly initialized.
WLREQUESTFAIL	Input request string format failure
WLINVALIDPARAMS	Invalid parameter(s) for function call
WLINVALIDRETURN	Invalid return designator for function call
WLFUNCTIONFAILED	RF device returned a failure code for the function
WLBCPARSEERROR	Failure parsing barcode file
WLBCADDERROR	Failed to add barcode to object
WLBCCLEARERROR	Failed to clear barcode list
WLBARCODELIMITEXCEEDED	Too many barcodes for configuration file
WLTONEADDERROR	Failed to add tone to object
WLTONECLEARERROR	Failed to clear tone list

WLIOQUEUEERROR	IO queue is empty
WLNOINPUTTYPE	Unable to determine input type
WLPORTTIMEOUT	Auxiliary port timeout
WLDTOUTOFSYNC	Terminal date/time out of sync by more than five seconds

See the Constant Values for the numeric equates returned by the function.

Example

See RFMenu Samples.

SetCoordinates Method

This member of RFMenu is supported on ALL devices.

The SetCoordinates method sets the placement and dimensions for an RFMenu object.

VB

```
object.SetCoordinates nColumn, nRow, nWidth, nHeight
```

VC++

```
HRESULT hr = object->SetCoordinates(short nColumn, short
nRow, short nWidth, short nHeight);
```

Parameters

nColumn	The on-screen column coordinate at which the top left corner of the menu will begin (starting with column 0).
nRow	The on-screen row coordinate at which the top left corner of the menu will begin (starting with row 0).
nWidth	The width of the entire menu, beginning from the coordinate given in the column parameter and measuring to the right.
nHeight	The height of the entire menu, beginning from the coordinate given in the row parameter and measuring downward.

Remarks

When defining the coordinates for a menu, keep in mind the limited size of an RF device's display. If the menu height and width are set to zero or not passed, WaveLink Studio COM will automatically scale the menu to either the size of the screen or the sum of the number of options and title lines (which ever is smaller).

Example

See the DoMenu Method.

SetMenuHeight Method

This member of RFMenu is supported on ALL devices.

The SetMenuHeight method defines the height of an RFMenu object.

VB

```
object.SetMenuHeight nHeight
```

VC++

```
HRESULT hr = object->SetMenuHeight(short nHeight);
```

Parameters

nHeight	The menu height, in characters, beginning from the row coordinate parameter of the RFMenu object and measuring downwards.

Remarks

When defining the coordinates for a menu, keep in mind the limited size of an RF device's display. If the menu height and width are set to zero or not passed, the WaveLink Studio COM will automatically scale the menu to either the size of the screen or the sum of the number of options and title lines (which ever is smaller).

Example

See the RFMenu Samples.

SetMenuWidth Method

This member of RFMenu is supported on ALL devices.

The SetMenuWidth method defines the width of the RFMenu object.

VB

```
object.SetMenuWidth nWidth
```

VC++

```
HRESULT hr = object->SetMenuWidth(short nWidth);
```

Parameters

nWidth	The menu width in characters, beginning from the column coordinate parameter of the RFMenu object and measuring to the right.

Remarks

When defining the coordinates for a menu, keep in mind the limited size of an RF device's display. If the height and width are set to zero or not passed, the WaveLink Studio COM will automatically scale the menu to either the size of the screen or the sum of the number of options and title lines (which ever is smaller).

Example

See the RFMenu Samples.

SetStartColumn Method

This member of RFMenu is supported on ALL devices.

The SetStartColumn method defines the starting column coordinate for an RFMenu object.

VB

```
object.SetStartColumn nColumn
```

VC++

```
HRESULT hr = object->SetStartColumn(short nColumn);
```

Parameters

nColumn	The column of the on-screen coordinate in which the top left corner of the menu should begin. Columns are numbered from the left starting with column 0.

Example

See the RFMenu Samples.

SetStartRow Method

This member of RFMenu is supported on ALL devices.

The SetStartRow method defines the starting row coordinate for an RFMenu object.

VB

```
object.SetStartRow nRow
```

VC++

```
HRESULT hr = object->SetStartRow(short nRow);
```

Parameters

nRow	The row of the on-screen coordinate in which the top left corner of the menu will begin. Rows are numbered from the top starting with row 0.

Example

See the RFMenu Samples.

StoreMenu Method

This member of RFMenu is supported on ALL devices.

The StoreMenu method saves the current RFMenu object as a menu file on an RF device for future use.

VB

```
bStatus = object.StoreMenu (pszMenuName)
```

VC++

```
HRESULT hr = object->StoreMenu(LPCTSTR pszMenuName, BOOL
*bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

pszMenuName	The file name in which the RFMenu object will be saved. This file name may be up to 8 characters in length.

Example

See the DoMenu Method.

RFMenu Samples

Using both Visual Basic and Visual C++ sample code , the following set of applications will demonstrate the implementation and capabilities of the RFMenu object and its methods. Both applications will store a menu with eight options on an RF device, return any user selection, then delete the menu from the device when the application is finished.

VB Example

```
' RFMenu Object Demo Application
'
' Import a sleep function from the kernel32 dll
Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)
' Declare an RFIO object for use throughout the
' demo application.
Public rfConsole As New RFIO
' Declare an RFMenu object for use throughout the
' demo application.
Public rfMnu As New RFMENU
' strBuffer is a temp storage buffer
Dim strBuffer As String
' bDone is a loop control variable
Dim bDone As Boolean
' nTmp is a temporary integer
Dim nTmp As Integer


Public Sub Main()
' This application demonstrates the use of the various RFMenu
' methods. It assumes a display area of 20x8. In practice, one
' should use an RFTerminal object to format output to fit the
' current device.
' The following lines clear the display and output the app title in
' reverse video.
    rfConsole.RFPrint 0, 0, "   Welcome To The   ", _
              WLCLEAR + WLREVERSE
    rfConsole.RFPrint 0, 1, "    RFMenu Demo!    ", WLREVERSE
    rfConsole.RFPrint 0, 3, "Hit a key to proceed", WLNORMAL
' Hide the cursor for a cleaner display
    rfConsole.RFPrint 0, 9, "", WLNORMAL
'
' Flush the output buffer and await a keystroke/scan
    If rfConsole.GetEvent() = "" Then
        GoTo ExitApp
    End If  ' If rfConsole.GetEvent() = ""
```

```
' Menus are useful for providing users with choices at run time
' without excessive RF traffic. During an application's
' initialization, one can store the static menus used by the app
' all at once. Then when needed, the application can execute them
' at will.

' To demonstrate their use, add some options to the RFMenu object ...
    rfMnu.ResetMenu           ' Not necessary with a new object,
                              ' but not a bad habit to acquire.
    rfMnu.AddOption "Option 1"
    rfMnu.AddOption "Option 2"
    rfMnu.AddOption "Option 3"
    rfMnu.AddOption "Option 4"
    rfMnu.AddOption "Option 5"
    rfMnu.AddOption "Option 6"
    rfMnu.AddOption "Option 7"
    rfMnu.AddOption "Option 8"

' and a title line or two ...
    rfMnu.AddTitleLine "RFMenu OLE Object"
    rfMnu.AddTitleLine "  Demonstration  "

' When storing the menu on the RF device, the library will place
' the menu in the upper left corner of the display and size it to
' height of the screen or height of the menu (options & title)
' if those values are set to 0. For this demo, we will be setting
' the values to see how the methods are used.
    rfMnu.SetStartColumn 0           ' The default value
    rfMnu.SetStartRow 1
    rfMnu.SetMenuWidth 20      ' The default value is the screen
                               ' width for the particular RF device
    rfMnu.SetMenuHeight 6      ' See above for discussion of this
                               ' value
' NOTE: One could set all these values using the SetCoordinates
' method:
'      rfMnu.SetCoordinates (0, 1, 20, 6);

' The menu must be stored on the RF device before it may be used.
    If rfMnu.StoreMenu("MenuDemo") = False Then
        GoTo ExitApp
    End If  ' If rfMnu.StoreMenu("MenuDemo") = False

' The following loop iterates while bDone == FALSE. It displays
' the stored menu, awaits the user's selection and displays
' the choice index and string value.
    bDone = False
```

```
    While bDone = False
        nTmp = rfMnu.DoMenu("MenuDemo")
        Select Case nTmp
        Case -2     ' An error has occurred
            strBuffer = "RFMenu Error : " + rfMnu.RFGetLastError()
            rfConsole.RFPrint 0, 3, strBuffer, WLCLEAR + WLNORMAL
            rfConsole.RFPrint 0, 7, "Hit a key to proceed", WLNORMAL
            ' Hide the cursor for a cleaner display
            rfConsole.RFPrint 0, 9, "", WLNORMAL
            rfConsole.GetEvent

        Case -1     ' CLR or CTL + 'X' was keyed
            bDone = True

        Case Else   ' User selected an option
            strBuffer = Str(nTmp) + " -> " + _
                        rfMnu.GetMenuOption(nTmp)
            rfConsole.RFPrint 0, 3, strBuffer, WLCLEAR + WLNORMAL
            rfConsole.RFPrint 0, 7, "Hit a key to proceed", WLNORMAL
            ' Hide the cursor for a cleaner display
            rfConsole.RFPrint 0, 9, "", WLNORMAL
            rfConsole.GetEvent
        End Select  ' Select Case nTmp
    Wend    ' While bDone = False

' Anytime an application changes the configuration of an RF device,
' it should return it to its original values during cleanup
    rfMnu.DeleteMenu "MenuDemo"

ExitApp:
End Sub
```

VC++ Example

```
////////////////////////////////////////////////////////
//    Place RF application's main code here
IRFIO       rfConsole;      // Declare an RFIO interface object
IRFMenu     rfMenu;         // Declare an RFMenu interface object
Cstring     csBuffer;       // Storage buffer for data strings
Short       nTmp;           // Temporary short value
BOOL        bDone;          // Denotes the user's desire to end
                            // the application

    // Create an RFIO object in the automated server for our use
    if (rfConsole.CreateDispatch ("WAVELINKOLE.RFIO") == FALSE) ↵
        return FALSE;
    // Create an RFMenu object in the automated server for our use
    if (rfMenu.CreateDispatch ("WAVELINKOLE.RFMENU") == FALSE) {
        // Release the RFIO object
        rfConsole.ReleaseDispatch ();
        return FALSE;
    } // if (rfTone.CreateDispatch ("WAVELINKOLE.RFMENU") == FALSE)

    // This application demonstrates the use of the various RFMenu
    // methods. It assumes a display area of 20x8. In practice, one
    // should use an RFTerminal object to format output to fit the
    // current device.
    // The following lines clear the display and output the app
    // title in reverse video.
    rfConsole.RFPrint (0, 0, "   Welcome To The   ", ↵
              WLCLEAR | WLREVERSE);
    rfConsole.RFPrint (0, 1, " RFMenu Method Demo ", WLREVERSE);
    rfConsole.RFPrint (0, 3, "Hit a key to proceed", WLNORMAL);
    // Hide the cursor for a cleaner display
    rfConsole.RFPrint (0, 9, "", WLNORMAL);

    // Flush the output buffer and await a keystroke/scan
    if ((rfConsole.GetEvent ()).IsEmpty () == TRUE) {
        // Release the RFIO & RFMenu objects
        rfConsole.ReleaseDispatch ();
        rfMenu.ReleaseDispatch ();
        return FALSE;
    } // if ((rfConsole.GetEvent ()).IsEmpty () == TRUE)

    // Menus are useful for providing users with choices at run
    // time without excessive RF traffic. During an application's
    // initialization, one can store the static menus used by the
    // app all at once. Then when needed, the application can
    // execute them at will.
```

```
// To demonstrate their use, add some options to the RFMenu
// object ...
rfMenu.ResetMenu (); // Not necessary with a new object,
                     // but not a
                     // bad habit to acquire.
rfMenu.AddOption ("Option 1");
rfMenu.AddOption ("Option 2");
rfMenu.AddOption ("Option 3");
rfMenu.AddOption ("Option 4");
rfMenu.AddOption ("Option 5");
rfMenu.AddOption ("Option 6");
rfMenu.AddOption ("Option 7");
rfMenu.AddOption ("Option 8");

// and a title line or two ...
rfMenu.AddTitleLine ("RFMenu OLE Object");
rfMenu.AddTitleLine ("  Demonstration  ");

// When storing the menu on the RF device, the library will
// place the menu in the upper left corner of the display and
// size it to the height of the screen or the height of the menu
// (options & title) if those values are set to 0. For
//  this demo, we will be setting the values to see how the
// methods are used.
rfMenu.SetStartColumn (0);     // The default value
rfMenu.SetStartRow (1);
rfMenu.SetMenuWidth (20);      // The default value is the
                               // screen width for the
                               // particular RF device
rfMenu.SetMenuHeight (6);      // See above for discussion of
                               // this value
// NOTE: One could set all these values using the
// SetCoordinates method:
//      rfMenu.SetCoordinates (0, 1, 20, 6);

// The menu must now be stored on the RF device before it
// may be used.
if (rfMenu.StoreMenu ("MenuDemo") == FALSE) {
    // Release the RFIO & RFMenu objects
    rfConsole.ReleaseDispatch ();
    rfMenu.ReleaseDispatch ();
    return FALSE;
} // if (rfMenu.StoreMenu ("MenuDemo") == FALSE)

// The following loop iterates while bDone == FALSE. It displays
```

```
// the stored menu, awaits the user's selection and displays the
// choice index and string value.
bDone = FALSE;
while (bDone == FALSE) {
    switch (nTmp = rfMenu.DoMenu ("MenuDemo")) {
    case -2 :      // An error has occurred
        csBuffer.Format ("RFMenu Error : %ld", ↵
                 rfMenu.RFRFGetLastError());
        rfConsole.RFPrint (0, 3, csBuffer, WLCLEAR | WLNORMAL);
        rfConsole.RFPrint (0, 7, "Hit a key to proceed", ↵
                  WLNORMAL);
        // Hide the cursor for a cleaner display
        rfConsole.RFPrint (0, 9, "", WLNORMAL);
        rfConsole.GetEvent ();

    case -1 :       // CLR or CTL + 'X' was keyed
        bDone = TRUE;
        break;

    default :       // User selected an option
        csBuffer.Format ("%d -> %s", nTmp, ↵
                 rfMenu.GetMenuOption (nTmp));
        rfConsole.RFPrint (0, 3, csBuffer, WLCLEAR | WLNORMAL);
        rfConsole.RFPrint (0, 7, "Hit a key to proceed", ↵
                  WLNORMAL);
        // Hide the cursor for a cleaner display
        rfConsole.RFPrint (0, 9, "", WLNORMAL);
        rfConsole.GetEvent ();
        break;
    } // switch (nTmp = rfMenu.DoMenu ("MenuDemo"))
} // while (bDone == FALSE)

// Anytime an application changes the configuration of an RF
// device, it should return it to its original values during
// cleanup
rfMenu.DeleteMenu ("MenuDemo");

// Release the RFIO & RFMenu objects
rfConsole.ReleaseDispatch ();
rfMenu.ReleaseDispatch ();
```

RFTerminal Object

The RFTerminal object is supported on ALL devices.

The RFTerminal object contains a variety of methods which provide current terminal state information and allow you to alter certain terminal options.

Methods

BackLight
CursorEnd
CursorMode
CursorStart
DiskSpace
KeyState
KeyTimeout
LithiumBattery
MainBattery
Memory
Ping
RawTerminalType
ReadTerminalInfo
RFGetLastError
SetBackLight
SetCursorEnd
SetCursorMode
SetCursorStart
SetDateTime
SetKeyState
SetKeyTimeout
SetPingCount
SetPingPacket
SetTerminalInfo
SystemCall
TerminalHeight
TerminalID
TerminalType
TerminalWidth
WaveLinkVersion

Prog IDs

"WAVELINKOLE.RFTERMINAL"

Remarks

An RFTerminal object contains the terminal settings. For network efficiency purposes, these settings are initially only set within the RFTerminal object when the object is first created. Therefore, to return the most current information you must first use the ReadTerminalInfo Method which will update the RFTerminal object with the latest device settings. For these same network efficiency reasons changes to device settings will not actually occur until a call to the SetTerminalInfo Method has been made.

Method Summary

BackLight Method
- Returns the amount of time that will pass before an RF device turns off the display backlight.

CursorEnd Method
- Returns the last line of device display that is valid for printing.

CursorMode Method
- Returns the current cursor type of the RF device.

CursorStart Method
- Returns the first line of device display that is valid for printing.

DiskSpace Method
- Returns the total storage capacity of the RF device.

KeyState Method
- Returns the state of the RF device's keypad.

KeyTimeout Method
- Returns the amount of inactive time that will pass before an RF device automatically powers down.

LithiumBattery Method
- Returns the state of the RF device's lithium battery.

MainBattery Method
- Returns the state of the RF device's main battery.

Memory Method
- Returns the total memory capacity of the RF device.

Ping Method
- Times a send and receive cycle between the RF application and the RF device.

RawTerminalType Method
- Returns the type of the current RF device.

ReadTerminalInfo Method
- Updates the RFTerminal object with the current device settings.

RFGetLastError Method
- Returns the value of the last generated RFTerminal error.

SetBackLight Method
- Defines the amount of time that will pass before an RF device turns off the display backlight.

SetCursorEnd Method
- Defines the last line of device display that is valid for printing.

SetCursorMode Method
- Defines the current cursor type of the RF device.

SetCursorStart Method
- Defines the first line of device display that is valid for printing.

SetDateTime Method
- Synchronizes the date and time on the RF device with the host time and date.

SetKeyState Method
- Sets the current state of the RF device's keypad.

SetKeyTimeout Method
- Sets the amount of time that will pass before the RF device automatically powers down.

SetPingCount Method
- Sets the number of ping cycles for the Ping method to execute.

SetPingPacket Method
- Sets the packet string sent by the Ping method.

SetTerminalInfo Method
- Applies altered device settings to the RF device.

SystemCall Method
- Executes common DOS commands on the RF device.

TerminalHeight Method
- Returns the height of the RF device's display.

TerminalID Method
- Returns the unique terminal ID of the RF device.

TerminalType Method
- Returns the RF device type.

TerminalWidth Method
- Returns the width of the RF device's display.

WaveLinkVersion Method
- Returns the version of the RF device's WaveLink Studio Client software.

BackLight Method

This member of RFTerminal is supported on ALL devices.

The BackLight method returns the amount of inactive time, in seconds, that may pass before an RF device automatically turns off the display backlight.

VB

```
nBackLight = object.BackLight ()
```

VC++

```
HRESULT hr = object->BackLight(short *nBackLight);
```

Return Value

nBackLight	The back light timeout variable. This method will return a -1 if an error occurs. Use the RFGetLastError method to return the generated error code.

Remarks

For network efficiency purposes, the settings within the RFTerminal object are initially only set when the object is first created. Therefore, to return the most current information you must first use the ReadTerminalInfo Method which will update the RFTerminal object with the latest device settings before making a call to any other methods.

CursorEnd Method

This member of RFTerminal is supported on ALL devices.

The CursorEnd method returns the last line of device display that is valid for printing.

VB

```
nEndLine = object.CursorEnd ()
```

VC++

```
HRESULT hr = object->CursorEnd(short *nEndLine);
```

Return Value

nEndLine	The last line of the RF device display that is valid for printing. Any line numbers that follow the returned line number may not be used to display text.

Remarks

For network efficiency purposes, the settings within the RFTerminal object are initially only set when the object is first created. Therefore, to return the most current information you must first use the ReadTerminalInfo Method which will update the RFTerminal object with the latest device settings before making a call to any other methods.

CursorMode Method

This member of RFTerminal is supported on ALL devices.

An RF device may use one of two different cursor types: a hardware based blinking block or a software based solid carrot. The CursorMode method will return the current state of the RF device's cursor.

VB

```
nCursorMode = object.CursorMode ()
```

VC++

```
HRESULT hr = object->CursorMode(short *nCursorMode);
```

Return Value

nCursorMode	The cursor state for the RF device (SOFTWARECURSOR or HARDWARECURSOR).

Remarks

The possible values are:

SOFTWARECURSOR	- Software solid carrot cursor.
HARDWARECURSOR	- Hardware blinking square cursor.

For network efficiency purposes, the settings within the RFTerminal object are initially only set when the object is first created. Therefore, to return the most current information you must first use the ReadTerminalInfo Method which will update the RFTerminal object with the latest device settings before making a call to any other methods.

CursorStart Method

This member of RFTerminal is supported on ALL devices.

The CursorStart method returns the first line of device display that is valid for printing.

VB

```
nStartLine = object.CursorStart ()
```

VC++

```
HRESULT hr = object->CursorStart(short *nStartLine);
```

Return Value

nStartLine	The first line of the RF device display that is valid for printing. Any line numbers that come prior to this line number may not be used to display text.

Remarks

For network efficiency purposes, the settings within the RFTerminal object are initially only set when the object is first created. Therefore, to return the most current information you must first use the ReadTerminalInfo Method which will update the RFTerminal object with the latest device settings before making a call to any other methods.

DiskSpace Method

This member of RFTerminal is supported on ALL devices.

The DiskSpace method returns the total storage capacity of the RF device. This method does not return the amount of storage space that is currently available.

VB

```
lSpace = object.DiskSpace ()
```

VC++

```
HRESULT hr = object->DiskSpace(long *lSpace);
```

Return Value

lSpace	The total storage capacity of the RF device.

Remarks

For network efficiency purposes, the settings within the RFTerminal object are initially only set when the object is first created. Therefore, to return the most current information you must first use the ReadTerminalInfo Method which will update the RFTerminal object with the latest device settings before making a call to any other methods.

KeyState Method

This member of RFTerminal is supported on ALL devices.

The KeyState method returns the state of the RF device's keypad.

VB

```
nKeyState = object.KeyState ()
```

VC++

```
HRESULT hr = object->KeyState(short *nKeyState);
```

Return Value

nKeyState	The device key state variable (CAPSLOCK or NORMALKEYS). This method will return a -1 if an error occurs. Use the RFGetLastError method to return the generated error code. The numeric equate values for the key state are as follows:

Remarks

The possible values are:

CAPSLOCK	The device's keypad is in caps lock mode.
NORMALKEYS	The device's keypad is in normal non-caps mode.

For network efficiency purposes, the settings within the RFTerminal object are initially only set when the object is first created. Therefore, to return the most current information you must first use the ReadTerminalInfo Method which will update the RFTerminal object with the latest device settings before making a call to any other methods.

KeyTimeout Method

This member of RFTerminal is supported on ALL devices.

The KeyTimeout method returns the amount of inactive time, in seconds, that may pass before an RF device automatically powers itself down.

VB

```
nTimeout = object.KeyTimeout ()
```

VC++

```
HRESULT hr = object->KeyTimeout(short *nTimeout);
```

Return Value

nTimeout	The device timeout variable. This method will return a -1 if an error occurs. Use the RFGetLastError method to return the generated error code.

Remarks

For network efficiency purposes, the settings within the RFTerminal object are initially only set when the object is first created. Therefore, to return the most current information you must first use the ReadTerminalInfo Method which will update the RFTerminal object with the latest device settings before making a call to any other methods.

LithiumBattery Method

This member of RFTerminal is supported on ALL devices.

The LithiumBattery method returns the state of the RF device's lithium battery.

VB

```
nLithiumState = object.LithiumBattery ()
```

VC++

```
HRESULT hr = object->LithiumBattery(short *nLithiumState);
```

Return Value

nLithiumState	The lithium battery state variable (see Remarks). This method will return a -1 if an error occurs. Use the RFGetLastError method to return the generated error code. The numeric equate values for the lithium battery state are as follows:

Remarks

The possible values are:

LITHIUMBATGOOD	The lithium battery is good.
LITHIUMBATDEAD	The lithium battery is dead.
LITHIUMBATNONE	The device does not have a lithium battery.

For network efficiency purposes, the settings within the RFTerminal object are initially only set when the object is first created. Therefore, to return the most current information you must first use the ReadTerminalInfo Method which will update the RFTerminal object with the latest device settings before making a call to any other methods.

MainBattery Method

This member of RFTerminal is supported on ALL devices.

The MainBattery method returns the state of the RF device's main battery.

VB

```
nBatState = object.MainBattery ()
```

VC++

```
HRESULT hr = object->MainBattery(short *nBatState);
```

Return Value

nBatState	The main battery state variable (MAINBATGOOD or MAINBATLOW). This method will return a -1 if an error occurs. Use the RFGetLastError method to return the generated error code.

Remarks

The possible values are:

MAINBATGOOD	- The main battery is good.
MAINBATLOW	- The main battery is low.

For network efficiency purposes, the settings within the RFTerminal object are initially only set when the object is first created. Therefore, to return the most current information you must first use the ReadTerminalInfo Method which will update the RFTerminal object with the latest device settings before making a call to any other methods.

Memory Method

This member of RFTerminal is supported on ALL devices.

The Memory method returns the total memory capacity of the RF device. This method does not return the amount of memory that is currently available.

VB

```
nMemory = object.Memory ()
```

VC++

```
HRESULT hr = object->Memory(long *nMemory);
```

Return Value

nMemory	The total memory capacity of the RF device.

Remarks

For network efficiency purposes, the settings within the RFTerminal object are initially only set when the object is first created. Therefore, to return the most current information you must first use the ReadTerminalInfo Method which will update the RFTerminal object with the latest device settings before making a call to any other methods.

Ping Method

This member of RFTerminal is supported on ALL devices.

The Ping Method times a send and receive cycle between the RF application and the RF device.

VB

```
lTime = object.Ping ()
```

VC++

```
HRESULT hr = object->Ping(double *lTime);
```

Return Value

lTime	The number of milliseconds for the ping packet round trip.

RawTerminalType Method

This member of RFTerminal is supported on ALL devices.

The RawTerminalType method returns the basic numeric values that are used to represent the various types of RF devices.

VB

```
nRawType = object.RawTerminalType ()
```

VC++

```
HRESULT hr = object->RawTerminalType(short *nRawType);
```

Return Value

nRawType	The numeric value used to represent the various types of RF devices (see Remarks). This method will return a -1 if an error occurs. Use RFGetLastError to return the generated error code.

Remarks

For network efficiency purposes, the settings within the RFTerminal object are initially only set when the object is first created. Therefore, to return the most current information you must first use the ReadTerminalInfo Method which will update the RFTerminal object with the latest device settings before making a call to any other methods.

The possible values for the device types are:

1 – LRT

2 – PDT

3 – PRC/VRC

4 – WS 10XX

5 – PDT 68XX

6 – PDT 61XX

11 – Palm Pilot

27 – CE, 2740

72 – CE, 7200

75 – CE, 7540

93 – Percon Falcon

5020 – Intermec Device

8000 – Windows Device

8001 – PPC

8002 – HPC

8003 – HPC Pro

Example

```
' VB Sample Code
Dim wlTerm As New RFTERMINAL
nRFTerminalType As Integer
.
.
.
  nRFTerminalType = wlterm.RawTerminalType

  ' The following code illustrates one method you may use in
  ' your applications to classify devices so
  ' that you can process their input differently later on.
  Select Case nRFTerminalType
    Case PDT1740
      nRFTerminalType = WIDGETDEVICE
    Case PDT2740
      nRFTerminalType = WIDGETDEVICE
  End Select
```

ReadTerminalInfo Method

This member of RFTerminal is supported on ALL devices.

For network efficiency purposes, the settings within the RFTerminal object are initially only set when the object is first created. The ReadTerminalInfo method updates the RFTerminal object with these values.

VB

```
bStatus = object.ReadTerminalInfo ()
```

VC++

```
HRESULT hr = object->ReadTerminalInfo(BOOL *bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Remarks

To update any changes you have made to an RF device, use the SetTerminalInfo Method.

Example

```
' VB Sample Code
Dim termIface As New RFTERMINAL
.
.
.
  ' request the current info from the device.
  termIface.ReadTerminalInfo
```

RFGetLastError Method

This member of RFTerminal is supported on ALL devices.

The RFGetLastError method returns the value of the last generated error.

VB

```
dError = object.RFGetLastError()
```

VC++

```
HRESULT hr = object->GetLastError(DWORD *dError);
```

Return Value

dError	The numeric DWORD value of the last generated function error.

Remarks

When ever a function error occurs, an error value is generated. You may use RFGetLastError to return the value of the error. The generated errors are as follows:

WLNOERROR	No error returned from function call.
WLCOMMERROR	RF network communication error
WLMEMORYERROR	Memory allocation error
WLNOTINITIALIZED	RF object is improperly initialized.
WLREQUESTFAIL	Input request string format failure
WLINVALIDPARAMS	Invalid parameter(s) for function call
WLINVALIDRETURN	Invalid return designator for function call
WLFUNCTIONFAILED	RF device returned a failure code for the function
WLBCPARSEERROR	Failure parsing barcode file
WLBCADDERROR	Failed to add barcode to object
WLBCCLEARERROR	Failed to clear barcode list
WLBARCODELIMITEXCEEDED	Too many barcodes for configuration file
WLTONEADDERROR	Failed to add tone to object
WLTONECLEARERROR	Failed to clear tone list

WLIOQUEUEERROR	IO queue is empty
WLNOINPUTTYPE	Unable to determine input type
WLPORTTIMEOUT	Auxiliary port timeout
WLDTOUTOFSYNC	Terminal date/time out of sync by more than five seconds

See the Constant Values for the numeric equates returned by the function.

SetBackLight Method

This member of RFTerminal is supported on ALL devices.

The SetBackLight method remotely sets the amount of inactive time, in seconds, that may pass before an RF device automatically turns off its display back light.

VB

```
bStatus = object.SetBackLight (nTimeout)
```

VC++

```
HRESULT hr = object->SetBackLight (short nTimeout, BOOL
*bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

nTimeout	The number of seconds of inactive time that may pass before an RF device automatically turns off the display back light.

Remarks

For network efficiency purposes, any changes made to an RF device will not take effect until the SetTerminalInfo Method is called. This allows you to change multiple options through the use of a single radio packet rather than using a radio packet for each option.

Note, if the backlight timeout value is more than the keytimeout value, the device will completely power itself down before the back light is ever deactivated.

SetCursorEnd Method

This member of RFTerminal is supported on ALL devices.

The SetCursorEnd method defines the last line of device display that is valid for printing.

VB

```
bStatus = object.SetCursorEnd (nEndLine)
```

VC++

```
HRESULT hr = object->SetCursorEnd (short nEndLine, BOOL
*bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

nEndLine	The last line of the RF device display that is valid for printing. For example, if you wanted to prevent printing to lines 6 and below of the RF device display, you would pass a 5 for this parameter, making line five the last valid display line.

Remarks

For network efficiency purposes, any changes made to an RF device will not take effect until the SetTerminalInfo Method is called. This allows you to change multiple options through the use of a single radio packet rather than using a radio packet for each option.

SetCursorMode Method

This member of RFTerminal is supported on ALL devices.

An RF device can use one of two different cursor types: a hardware based blinking block or a software based solid carrot. The SetCursorMode method remotely sets the state of an RF device's cursor.

VB

```
bStatus = object.SetCursorMode (nCursor)
```

VC++

```
HRESULT hr = object->SetCursorMode (short nCursor, BOOL
*bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

nCursor	The cursor state for the RF device (HARDWARECURSOR or SOFTWARECURSOR).

Remarks

The possible values are:

HARDWARECURSOR	- Hardware blinking block cursor.
SOFTWARECURSOR	- Software v-type cursor.

For network efficiency purposes, any changes made to an RF device will not take effect until the SetTerminalInfo Method is called. This allows you to change multiple options through the use of a single radio packet rather than using a radio packet for each option.

Example

See the RFTerminal Samples.

SetCursorStart Method

This member of RFTerminal is supported on ALL devices.

The SetCursorStart method defines the first line of device display that is valid for printing.

VB

```
bStatus = object.SetCursorStart (nStartLine)
```

VC++

```
HRESULT hr = object->SetCursorStart (short nStartLine, BOOL
*bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

nStartLine	The first line of the RF device display that is valid for printing. For example, if you wanted to prevent printing to lines 3 and above of the RF device display, you would pass a 4 for this parameter, making line four the first valid display line.

Remarks

For network efficiency purposes, any changes made to an RF device will not take effect until the SetTerminalInfo Method is called. This allows you to change multiple options through the use of a single radio packet rather than using a radio packet for each option.

SetDateTime Method

This member of RFTerminal is supported on ALL devices.

The SetDateTime method synchronizes the date and time on the RF device with the host by sending it the current host date and time.

VB

```
bStatus = object.SetDateTime()
```

VC++

```
HRESULT hr = object->SetDateTime(BOOL *bStatus);
```

Parameters

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

SetKeyState Method

This member of RFTerminal is supported on ALL devices.

The SetKeyState method remotely sets the current state of the RF device's keypad.

VB

```
bStatus = object.SetKeyState (nKeyState)
```

VC++

```
HRESULT hr = object->SetKeyState (short nKeyState, BOOL
*bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

nKeyState	The key state for the RF device's keypad (CAPSLOCK or NORMALKEYS).

Remarks

The possible values are:

CAPSLOCK	The device's keypad is in caps lock mode.
NORMALKEYS	The device's keypad is in normal non-caps mode.

For network efficiency purposes, any changes made to an RF device will not take effect until the SetTerminalInfo Method is called. This allows you to change multiple options through the use of a single radio packet rather than using a radio packet for each option.

SetKeyTimeout Method

This member of RFTerminal is supported on ALL devices.

The SetKeyTimeout method remotely sets the amount of inactive time, in seconds, that may pass before an RF device automatically shuts itself down.

VB

```
bStatus = object.SetKeyTimeout (nTimeout)
```

VC++

```
HRESULT hr = object->SetKeyTimeout (short nTimeout, BOOL
*bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

nTimeout	The number of seconds of inactive time that may pass before an RF device automatically shuts itself down.

Remarks

For network efficiency purposes, any changes made to an RF device will not take effect until the SetTerminalInfo Method is called. This allows you to change multiple options through the use of a single radio packet rather than using a radio packet for each option.

SetPingCount Method

This member of RFTerminal is supported on ALL devices.

The SetPingCount method sets the number of ping cycles for the Ping method to execute.

VB

```
bStatus = object.SetPingCount (lPingCount)
```

VC++

```
HRESULT hr = object->SetPingCount (long lPingCount, BOOL
*bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

lPingCount	The number of ping cycles to execute per Ping command.

SetPingPacket Method

This member of RFTerminal is supported on ALL devices.

The SetPingPacket method sets the packet string sent by the Ping method.

VB

```
bStatus = object.SetPingPacket (pszPingPacket)
```

VC++

```
HRESULT hr = object->SetPingPacket (LPCTSTR pszPingPacket,
BOOL *bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

pszPingPacket	The packet string to be sent by the Ping method.

SetTerminalInfo Method

This member of RFTerminal is supported on ALL devices.

For network efficiency purposes, any changes made to an RF device settings will not take effect until the SetTerminalInfo method is called. This way, multiple device settings may be altered through a single RF packet, reducing network traffic, rather than sending a number of packets for each change.

VB

```
bStatus = object.SetTerminalInfo ()
```

VC++

```
HRESULT hr = object->SetTerminalInfo(BOOL *bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Remarks

To update the RFTerminal object with the most current device settings, use the ReadTerminalInfo Method.

Example

See the RFTerminal Samples.

SystemCall Method

This member of RFTerminal is supported on ALL devices.

The SystemCall method lets you execute common DOS commands and functions on an RF device.

VB

```
bStatus = object.SystemCall (pszSystemCall)
```

VC++

```
HRESULT hr = object->SystemCall (LPCTSTR pszSystemCall, BOOL
*bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

pszSystemCall	The DOS command you wish to execute. For example, for a directory listing of the RF device, simply pass "DIR".

TerminalHeight Method

This member of RFTerminal is supported on ALL devices.

The TerminalHeight method returns the maximum height, in characters, of an RF device's display.

VB

```
nHeight = object.TerminalHeight ()
```

VC++

```
HRESULT hr = object->TerminalHeight(short *nHeight);
```

Return Value

nHeight	The terminal display height variable.

Remarks

For network efficiency purposes, the settings within the RFTerminal object are initially only set when the object is first created. Therefore, to return the most current information you must first use the ReadTerminalInfo Method which will update the RFTerminal object with the latest device settings before making a call to any other methods.

Example

```
' VB Sample Code
Dim termIface As New RFTERMINAL
wlio AS New RFIO
termWidth As Integer
termHeight As Integer
.
.
.
  ' if you need to obtain current terminal info, use the
  ' ReadTerminalInfo method (not shown).
  termWidth = termIface.TerminalWidth
  termHeight = termIface.TerminalHeight
.
.
.
  ' In this example, use the variables with an RFPrint call.
  wlio.RFPrint 0, (termHeight - 1), "Correct y/n? ", WLNORMAL
  ' hide the cursor for a cleaner display using TermWidth or
  ' TermHeight
  wlio.RFPrint 0, (termWidth + 1), "", WLNORMAL
```

TerminalID Method

This member of RFTerminal is supported on ALL devices.

For identification purposes, each RF device in a WaveLink network is assigned a unique terminal ID. The TerminalID method returns the RF device terminal ID.

VB

```
pszTerminalID = object.TerminalID ()
```

VC++

```
HRESULT hr = object->TerminalID(Cstring *pszTerminalID);
```

Return Value

pszTerminalID The device's Terminal ID.

Remarks

For network efficiency purposes, the settings within the RFTerminal object are initially only set when the object is first created. Therefore, to return the most current information you must first use the ReadTerminalInfo Method which will update the RFTerminal object with the latest device settings before making a call to any other methods.

Note:For Spectrum24 networks the RF device's IP address is its Terminal ID.

Example

See the RFTerminal Samples.

TerminalType Method

This member of RFTerminal is supported on ALL devices.

The TerminalType method returns the specific type of RF device.

VB

```
pszDevType = object.TerminalType ()
```

VC++

```
HRESULT hr = object->TerminalType(Cstring *pszDevType);
```

Return Value

pszDevType	The terminal type variable (see Remarks). If this method returns an empty string, an error may have occurred. Use the RFGetLastError method to return the generated error code.

Remarks

The possible values are:

LRT	- Laser Radio Terminal
PDT	- Portable Data Terminal
PRC	- Portable Radio Computer
VRC	- Vehicular Radio Computer
WS	- Wearable System Terminal
Palm Pilot	- Palm Pilot
CE – 2740	- Model 2740
CE – 7200	- Model 7200
CE – 7540	- Model 7540
Percon Falcon	- Percon Falcon
Intermec Device	- Intermec Device
Windows Device	- Windows Device
PPC	- Palm PC
HPC	- Handheld PC
HPC Pro	- HPC Pro

For network efficiency purposes, the settings within the RFTerminal object are initially only set when the object is first created. Therefore, to return the most current information you must first use the ReadTerminalInfo Method which will update the RFTerminal object with the latest device settings before making a call to any other methods.

TerminalWidth Method

This member of RFTerminal is supported on ALL devices.

The TerminalWidth method will return the maximum width in characters of an RF device's display.

VB

```
nWidth = object.TerminalWidth ()
```

VC++

```
HRESULT hr = object->TerminalWidth(short *nWidth);
```

Return Value

nWidth	The terminal display width variable. This method will return a -1 if an error occurs. Use the RFGetLastError method to return the generated error message.

Remarks

For network efficiency purposes, the settings within the RFTerminal object are initially only set when the object is first created. Therefore, to return the most current information you must first use the ReadTerminalInfo Method which will update the RFTerminal object with the latest device settings before making a call to any other methods.

Example

See the TerminalHeight Method.

WaveLinkVersion Method

This member of RFTerminal is supported on ALL devices.

The WaveLinkVersion method returns the version number of the RF device's WaveLink Studio Client.

VB

```
pszVersion = object.WaveLinkVersion ()
```

VC++

```
HRESULT hr = object->WaveLinkVersion(Cstring *pszVersion);
```

Return Value

pszVersion	The version number of the RF device's WaveLink Client.

Remarks

For network efficiency purposes, the settings within the RFTerminal object are initially only set when the object is first created. Therefore, to return the most current information you must first use the ReadTerminalInfo Method which will update the RFTerminal object with the latest device settings before making a call to any other methods.

RFTerminal Samples

Using both Visual Basic and Visual C++ sample code, the following set of applications will demonstrate the implementation and capabilities of the RFTerminal object and its methods. Both applications will return the values for various settings of an RF device, such as display width and height, and temporarily alter some of these settings.

VB Example

```
' RFTerminal Object Demo Application
'
' Import a sleep function from the kernel32 dll
Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)
' Declare an RFIO object for use throughout the
' demo application.
Public rfConsole As New RFIO
' Declare an RFTerminal object for use throughout the
' demo application.
Public rfTerm As New RFTERMINAL
' strBuffer is a temp storage buffer
Dim strBuffer As String


Public Sub Main()
' This application demonstrates the use of the various RFTerminal meth
ods.
' The following lines clear the display and output the app title in
' reverse video.
    rfConsole.RFPrint 0, 0, "   Welcome To The   ", _
              WLCLEAR + WLREVERSE
    rfConsole.RFPrint 0, 1, "  RFTerminal Demo!  ", WLREVERSE
    rfConsole.RFPrint 0, 3, "Hit a key to proceed", WLNORMAL
' Hide the cursor for a cleaner display
    rfConsole.RFPrint 0, 9, "", WLNORMAL
'
' Flush the output buffer and await a keystroke/scan
    If rfConsole.GetEvent() = "" Then
        GoTo ExitApp
    End If  ' If rfConsole.GetEvent() = ""

' When designing RF applications, it is often not possible to know
' which types of devices are going to be used by the end user. Using
' an RFTerminal object allows you access to accurate information
' about the user's RF device. Simply creating an RFTerminal object
' acquires the terminal's information, but for demonstration
```

```
' purposes, we will explicitly request the current setting from
' the RF device.
    If rfTerm.ReadTerminalInfo() = False Then
        GoTo ExitApp
    End If  ' If rfTerm.ReadTerminalInfo() = False
    rfConsole.RFPrint 0, 0, "Terminal Information", _
              WLREVERSE + WLCLEAR

' Display the terminal's screen dimensions
    strBuffer = "Screen Size (" + Str(rfTerm.TerminalWidth()) + _
        "," + Str(rfTerm.TerminalHeight()) + ")"
    rfConsole.RFPrint 0, 1, strBuffer, WLNORMAL

' Display the WaveLink Studio Client version
    rfConsole.RFPrint 0, 3, "WaveLink Version:", WLNORMAL
    rfConsole.RFPrint 0, 4, rfTerm.WaveLinkVersion(), WLNORMAL

' Display the Terminal's ID
    rfConsole.RFPrint 0, 6, "Terminal ID:", WLNORMAL
    rfConsole.RFPrint 0, 7, rfTerm.TerminalID(), WLNORMAL

' Hide the cursor for a cleaner display. Here we use the screen
' height method to place our cursor off the viewable area.
    rfConsole.RFPrint 0, rfTerm.TerminalHeight() + 1, "", WLNORMAL

' Flush the output buffer and await a keystroke/scan
    If rfConsole.GetEvent() = "" Then
        GoTo ExitApp
    End If  ' If rfConsole.GetEvent() = ""

    rfConsole.RFPrint 0, 0, " Setting Cursor To  ", _
              WLCLEAR + WLREVERSE
    rfConsole.RFPrint 0, 1, "   Software Mode    ", WLREVERSE
    rfConsole.RFPrint 0, 3, "SW Cursor -> ", WLFLUSHOUTPUT
    rfTerm.SetCursorMode SOFTWARECURSOR
    If rfTerm.SetTerminalInfo() = False Then
        GoTo ExitApp
    End If  ' If rfTerm.SetTerminalInfo() = False
    Sleep (2000)

    rfConsole.RFPrint 0, 5, "Resetting Cursor To ", WLREVERSE
    rfConsole.RFPrint 0, 6, "  Hardware Mode     ", _
              WLREVERSE + WLFLUSHOUTPUT
    rfTerm.SetCursorMode HARDWARECURSOR
    If rfTerm.SetTerminalInfo() = False Then
        GoTo ExitApp
```

```
    End If  ' If rfTerm.SetTerminalInfo() = False
    rfConsole.RFPrint 0, 7, "Done.", WLNORMAL

' Hide the cursor for a cleaner display. Here we use the screen
' height method to place our cursor off the viewable area.
    rfConsole.RFPrint 0, rfTerm.TerminalHeight() + 1, "", WLNORMAL

' Flush the output buffer and await a keystroke/scan
    If rfConsole.GetEvent() = "" Then
        GoTo ExitApp
    End If  ' If rfConsole.GetEvent() = ""

ExitApp:
End Sub
```

VC++ Example

```
////////////////////////////////////////////////////////
//       Place RF application's main code here
IRFIO          rfConsole;       // Declare an RFIO interface object
IRFTerminal    rfTerminal;      // Declare an RFTerminal interface
                                // object
Cstring        csBuffer;        // Storage buffer for data strings

    // Create an RFIO object in the automated server for our use

    if (rfConsole.CreateDispatch ("WAVELINKOLE.RFIO") == FALSE) ↵
        return FALSE;
    // Create an RFTerminal object in the automated server for
    // our use
    if (rfTerminal.CreateDispatch ("WAVELINKOLE.RFTERMINAL") == ↵
         FALSE) return FALSE;

    // This application demonstrates the use of the various
    // RFTerminal methods.
    // The following lines clear the display and output the app
    // title in reverse video.
    rfConsole.RFPrint (0, 0, "   Welcome To The   ", ↵
              WLCLEAR | WLREVERSE);
    rfConsole.RFPrint (0, 1, "  RFTerminal Demo!  ", WLREVERSE);
    rfConsole.RFPrint (0, 3, "Hit a key to proceed", WLNORMAL);
    // Hide the cursor for a cleaner display
    rfConsole.RFPrint (0, 9, "", WLNORMAL);

    // Flush the output buffer and await a keystroke/scan
    if ((rfConsole.GetEvent ()).IsEmpty () == TRUE) {
        // Release the RFIO & RFTerminal objects
        rfConsole.ReleaseDispatch ();
        rfTerminal.ReleaseDispatch ();
        return FALSE;
    } // if ((rfConsole.GetEvent ()).IsEmpty () == TRUE)

    // When designing RF applications, it is often impossible to
    // know which types of devices will be used by the end user.
    // Using an RFTerminal object allows you access to accurate
    // information about the user's RF device. Simply creating an
    // RFTerminal object acquires the terminal's information, but
    // for demonstration purposes, we will explicitly request
    // the current setting from the RF device.
    if (rfTerminal.ReadTerminalInfo () == FALSE) {
        // Release the RFIO & RFTerminal objects
        rfConsole.ReleaseDispatch ();
```

```
    rfTerminal.ReleaseDispatch ();
    return FALSE;
} // if (rfTerminal.ReadTerminalInfo () == FALSE)

rfConsole.RFPrint (0, 0, "Terminal Information", ↵
          WLREVERSE | WLCLEAR);

// Display the terminal's screen dimensions
csBuffer.Format ("Screen Size (%d,%d)", ↵
         rfTerminal.TerminalWidth (), ↵
         rfTerminal.TerminalHeight ());
rfConsole.RFPrint (0, 1, csBuffer, WLNORMAL);

// Display the WaveLink Studio Client version
rfConsole.RFPrint (0, 3, "WaveLink Version:", WLNORMAL);
rfConsole.RFPrint (0, 4, rfTerminal.WaveLinkVersion (), ↵
          WLNORMAL);

// Display the Terminal's ID
rfConsole.RFPrint (0, 6, "Terminal ID:", WLNORMAL);
rfConsole.RFPrint (0, 7, rfTerminal.TerminalID (), WLNORMAL);

// Hide the cursor for a cleaner display. Here we use the
// screen height method to place
// our cursor off the viewable area.
rfConsole.RFPrint (0, rfTerminal.TerminalHeight () + 1, "", ↵
          WLNORMAL);

// Flush the output buffer and await a keystroke/scan
if ((rfConsole.GetEvent ()).IsEmpty () == TRUE) {
    // Release the RFIO & RFTerminal objects
    rfConsole.ReleaseDispatch ();
    rfTerminal.ReleaseDispatch ();
    return FALSE;
} // if ((rfConsole.GetEvent ()).IsEmpty () == TRUE)

rfConsole.RFPrint (0, 0, " Setting Cursor To  ", ↵
          WLCLEAR | WLREVERSE);
rfConsole.RFPrint (0, 1, "   Software Mode    ", WLREVERSE);
rfConsole.RFPrint (0, 3, "SW Cursor -> ", WLFLUSHOUTPUT);
rfTerminal.SetCursorMode (SOFTWARECURSOR);
if (rfTerminal.SetTerminalInfo () == FALSE) {
    // Release the RFIO & RFTerminal objects
    rfConsole.ReleaseDispatch ();
    rfTerminal.ReleaseDispatch ();
    return FALSE;
```

```
} // if (rfTerminal.SetTerminalInfo () == FALSE)
else Sleep (2000);

rfConsole.RFPrint (0, 5, "Resetting Cursor To ", WLREVERSE);
rfConsole.RFPrint (0, 6, "  Hardware Mode     ", ↵
          WLREVERSE | WLFLUSHOUTPUT);
rfTerminal.SetCursorMode (HARDWARECURSOR);
if (rfTerminal.SetTerminalInfo () == FALSE) {
    // Release the RFIO & RFTerminal objects
    rfConsole.ReleaseDispatch ();
    rfTerminal.ReleaseDispatch ();
    return FALSE;
} // if (rfTerminal.SetTerminalInfo () == FALSE)
rfConsole.RFPrint (0, 7, "Done.", WLNORMAL);

// Hide the cursor for a cleaner display. Here we use
// the screen height method to place
// our cursor off the viewable area.
rfConsole.RFPrint (0, rfTerminal.TerminalHeight () + 1, "", ↵
          WLNORMAL);

// Flush the output buffer and await a keystroke/scan
    if ((rfConsole.GetEvent ()).IsEmpty () == TRUE) {
    // Release the RFIO & RFTerminal objects
    rfConsole.ReleaseDispatch ();
    rfTerminal.ReleaseDispatch ();
    return FALSE;
} // if ((rfConsole.GetEvent ()).IsEmpty () == TRUE)

// Release the RFIO & RFTerminal objects
rfConsole.ReleaseDispatch ();
rfTerminal.ReleaseDispatch ();
```

RFTone Object

The RFTone object is supported on ALL devices.

The RFTone object lets you utilize the audio capabilities of a remote RF device from your applications. Using an RFTone object you can compose multiple notes into single tone files, save these files to an RF device, then play the tones.

Methods

AddTone
ClearTones
DeleteToneFile
GetDuration
GetFrequency
GetToneFile
ListToneFiles
PlayTone
RemoveTone
RFGetLastError
SetDuration
SetFrequency
StoreTone
ToneCount
ToneFileCount
ToneFileName

Prog IDs

"WAVELINKOLE.RFTONE"

Remarks

Individual notes are stored within an RFTone object, each note defined by a frequency and a duration. Once defined, a specific note will remain within an RFTone object until either the object is released or all notes are cleared from the object using the ClearTones Method. This allows you to use a single tone object within your application which may be modified rather than creating a new tone object for each use.

Method Summary

AddTone Method
- Adds a specific note to the RFTone object.

ClearTones Method
- Clears the notes from the RFTone object.

DeleteToneFile Method
- Deletes tone files from the RF device.

GetDuration Method
- Returns the duration of a specific note in the RFTone object.

GetFrequency Method
- Returns the frequency of a specific note in the RFTone object.

GetToneFile Method
- Returns the a tone file into the current RFTone object.

ListToneFiles Method
- Returns the number of tone files on the Rfdevice and stores the list of names in the RFTone object.

PlayTone Method
- Plays a tone file.

RemoveTone Method
- Removes a specific note from the current RFTone object.

RFGetLastError Method
- Returns the value of the last generated RFTone error.

SetDuration Method
- Defines the duration of a specific note in the RFTone object.

SetFrequency Method
- Defines the frequency of a specific note in the RFTone object.

StoreTone Method
- Stores the RFTone object as a tone file on the RF device for future use.

ToneCount Method
- Returns the number of notes within the RFTone object.

ToneFileCount Method
- Returns the number of tone files returned by the last successful call to the ListToneFiles method.

ToneFileName Method
- Returns the file name of a saved tone file.

AddTone Method

This member of RFTone is supported on ALL devices.

The AddTone method lets you add a specific note to an RFTone object.

VB

```
bStatus = object.AddTone (nFrequency, nDuration)
```

VC++

```
HRESULT hr = object->AddTone(short nFrequency, short
nDuration, BOOL *bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

nFrequency	The frequency for a single tone defined in numeric hertz (cycles per second).
nDuration	The duration of time the tone will play measured in milliseconds.

Remarks

Individual notes are stored within an RFTone object, each note defined by a frequency and a duration. Once defined, a specific note will remain within an RFTone object until either the object is released or all notes are cleared from the object using the ClearTones Method. This allows you to use a single tone object within your application that may be modified rather than creating a new tone object for each use.

Example

See the PlayTone Method.

ClearTones Method

This member of RFTone is supported on ALL devices.

The ClearTones method lets you completely clear an RFTone object of any notes.

VB

```
bStatus = object.ClearTones ()
```

VC++

```
HRESULT hr = object->ClearTones(BOOL *bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Remarks

Individual notes are stored within an RFTone object, each note defined by a frequency and a duration. Once defined, a specific note will remain within an RFTone object until either the object is released or all notes are cleared from the object using the ClearTones Method. This allows you to use a single tone object within your application that may be modified rather than creating a new tone object for each use.

Example

See the PlayTone Method.

DeleteToneFile Method

This member of RFTone is supported on ALL devices.

The DeleteToneFile method lets you remove tone files from an RF device's non-volatile memory.

VB

```
bStatus = object.DeleteToneFile (pszFileName)
```

VC++

```
HRESULT hr = object->DeleteToneFile(LPCTSTR pszFileName,
BOOL *bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

pszFileName	The name of the tone file stored on the RF device that you wish to delete. You may pass a single eight character file name or enter a wildcard symbol ("*") to delete all tone files from an RF device.

Example

See the RFTone Samples.

GetDuration Method

This member of RFTone is supported on ALL devices.

The GetDuration method returns the duration for a specific note in the current RFTone object, measured in cycles per second.

VB

```
nDuration = object.GetDuration (nTone)
```

VC++

```
HRESULT hr = object->GetDuration(short nTone, short
*nDuration);
```

Return Value

nDuration	The duration of the note in milliseconds.

Parameters

nTone	The zero based index value note within the RFTone object that's duration is being returned. Remember, the first note is zero. For example, to return the frequency for the third note in the current RFTone object you would pass a 2

Remarks

To return the duration for a note saved within a tone file on an RF device, first set the saved tone file as the current RFTone object using the GetToneFile Method, then use GetDuration.

GetFrequency Method

This member of RFTone is supported on ALL devices.

The GetFrequency method returns the frequency of a specific tone in the current RFTone object.

VB

```
nFreq = object.GetFrequency (nTone)
```

VC++

```
HRESULT hr = object->GetFrequency(short nTone, short
*nFreq);
```

Return Value

nFreq	The variable containing the returned frequency value. This function will return a -1 if an error occurs. Use RFGetLastError to return the generated error code.

Parameters

nTone	The index value for the note that's frequency is being returned. Remember, the first note is zero. For example, to return the frequency for the third note in the current RFTone object you would pass a 2.

Remarks

To return a note frequency stored within a tone file on an RF device, first use the GetToneFile Method to set the tone file as the current tone object within the application, then use GetFrequency.

GetToneFile Method

This member of RFTone is supported on ALL devices.

The GetToneFile method retrieves a tone file from an RF device into the current RFTone object within your application.

VB

```
nStatus = object.GetToneFile (pszFileName)
```

VC++

```
HRESULT hr = object->GetToneFile(LPCTSTR pszFileName, short
*nStatus);
```

Return Value

nStatus	The total number of notes placed within the current RFTone object. This method will return a -1 if an error occurs. Use the RFGetLastError method to return the error value. If the RFTone file does not exist on the RF device, this method will return 0.

Parameters

pszFileName	The name of the tone file that you wish to set as the current RFTone object.

Example

See the RFTone Samples.

ListToneFiles Method

This member of RFTone is supported on ALL devices.

The ListToneFiles method returns the total number of tone files saved on an RF device and stores a list of the file names in the current object.

VB

```
nTones = object.ListToneFiles ()
```

VC++

```
HRESULT hr = object->ListToneFiles(short *nTones);
```

Return Value

nTones	The name of the variable for the returned tone file count. This function will return a -1 if an error occurs. Use the RFGetLastError method to return the generated error code.

Remarks

To return the value of the last successful ListToneFiles method without making an actual call to the RF device, use the ToneFileCount Method. To list the total number of notes within a single tone file, use the ToneCount Method.

PlayTone Method

This member of RFTone is supported on ALL devices.

The PlayTone method plays a tone file saved on an RF device.

VB

```
bStatus = object.PlayTone (pszFileName)
```

VC++

```
HRESULT hr = object->PlayTone(LPCTSTR pszFileName, BOOL
*bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

pszFileName	The name of the tone file stored on the RF device that you wish to play.

Remarks

RFTone objects must be saved as tone files on an RF device using the StoreTone Method BEFORE they may be played.

Example

```
' VB Sample Code
Dim wlTone As New RFTONE.
.
.
  wlTone.ClearTones
  wlTone.AddTone 440, 200
  wlTone.AddTone 440, 200
  If wlTone.StoreTones ("ErrorBeep") = False Then
    GoTo ExitApp
  End If
.
.
.
  ' play the tone
  wlTone.PlayTone "ErrorBeep"
```

RemoveTone Method

This member of RFTone is supported on ALL devices.

The RemoveTone method lets you remove a specific note from the current RFTone object.

VB

```
bStatus = object.RemoveTone (nTone)
```

VC++

```
HRESULT hr = object->RemoveTone(short nTone, BOOL *bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

nTone	The zero based index value note to be removed. Note, the first tone in the list is indexed as zero. For example, to remove the frequency for the third note in the current RFTone object you would pass a 2.

Remarks

To remove a note from a tone file stored on an RF device, you must first set the tone file as the current RFTone object using the GetToneFile Method, then use RemoveTone.

RFGetLastError Method

This member of RFTone is supported on ALL devices.

The RFGetLastError method returns the value of the last generated error.

VB

```
dError = object.RFGetLastError()
```

VC++

```
HRESULT hr = object->GetLastError(DWORD *dError);
```

Return Value

dError	The numeric DWORD value of the last generated function error.

Remarks

When ever a function error occurs, an error value is generated. You may use RFGetLastError to return the value of the error. The generated errors are as follows:

WLNOERROR	No error returned from function call.
WLCOMMERROR	RF network communication error
WLMEMORYERROR	Memory allocation error
WLNOTINITIALIZED	RF object is improperly initialized.
WLREQUESTFAIL	Input request string format failure
WLINVALIDPARAMS	Invalid parameter(s) for function call
WLINVALIDRETURN	Invalid return designator for function call
WLFUNCTIONFAILED	RF device returned a failure code for the function
WLBCPARSEERROR	Failure parsing barcode file
WLBCADDERROR	Failed to add barcode to object
WLBCCLEARERROR	Failed to clear barcode list
WLBARCODELIMITEXCEEDED	Too many barcodes for configuration file
WLTONEADDERROR	Failed to add tone to object
WLTONECLEARERROR	Failed to clear tone list

WLIOQUEUEERROR	IO queue is empty
WLNOINPUTTYPE	Unable to determine input type
WLPORTTIMEOUT	Auxiliary port timeout
WLDTOUTOFSYNC	Terminal date/time out of sync by more than five seconds

See the Constant Values for the numeric equates returned by the function.

Example

See the RFTone Samples.

SetDuration Method

This member of RFTone is supported on ALL devices.

The SetDuration method lets you alter the duration for an existing note within an RFTone object.

VB

```
bStatus = object.SetDuration (nDuration, nTone)
```

VC++

```
HRESULT hr = object->SetDuration(short nDuration, short
nTone, BOOL *bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

nDuration	The new duration for the selected note, measured in milliseconds.
nTone	The zero based index value note whose duration is to be altered. Remember, the first note in the list is indexed as zero. For example, to alter the duration for the third note in the current RFTone object you would pass a 2.

Remarks

To alter the duration for a note stored within a tone file on an RF device, first use the GetToneFile Method to set the tone file as the current tone object within the application, then use SetDuration.

Example

See the RFTone Samples.

SetFrequency Method

This member of RFTone is supported on ALL devices.

The SetFrequency method lets you alter the frequency for an existing tone within an RFTone object.

VB

```
bStatus = object.SetFrequency (nFrequency, nTone)
```

VC++

```
HRESULT hr = object->SetFrequency(short nFrequency, short
nTone, BOOL *bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

nFrequency	The new frequency for the existing tone, as defined in numeric hertz.
nTone	The zero based index value note that's frequency is being altered. Note, the first note in the list is indexed as zero. For example, to alter the frequency for the third note in the current RFTone object you would pass 2.

Remarks

To alter a frequency for a note stored within a tone file on an RF device, first use the GetToneFile Method to set the tone file as the current tone object within the application, then use SetFrequency.

StoreTone Method

This member of RFTone is supported on ALL devices.

The StoreTone method lets you save an RFTone object directly to an RF device's non-volatile memory for future use.

VB

```
bStatus = object.StoreTone (pszFileName)
```

VC++

```
HRESULT hr = object->StoreTone(LPCTSTR pszFileName, BOOL
*bStatus);
```

Return Value

bStatus	The status of the function returned as Boolean True or False values. Use the RFGetLastError method to return the error value.

Parameters

pszFileName	The file name that the current RFTone object will be saved as on the RF device. This file name may be up to 8 characters in length.

Remarks

RFTone objects must be saved as tone files on an RF device using the StoreTone method BEFORE they may be played. Use the PlayTone method to play a tone.

Example

See the PlayTone Method.

ToneCount Method

This member of RFTone is supported on ALL devices.

The ToneCount method returns the number of individual notes within an RFTone object.

VB

```
nNotes = object.ToneCount ()
```

VC++

```
HRESULT hr = object->ToneCount(short *nNotes);
```

Return Value

nNotes	The name of the variable for the returned note count. This function will return a -1 if an error occurs. Use RFGetLastError to return the generated error code.

Remarks

To count the number of notes within a tone file saved to an RF device, first set the tone file as the current RFTone object using the GetToneFile Method, then use the ToneCount method.

Example

See the RFTone Samples.

ToneFileCount Method

This member of RFTone is supported on ALL devices.

The ToneFileCount method returns the total number of tone files returned by the last successful call to the ListToneFiles method.

VB

```
nFiles = object.ToneFileCount ()
```

VC++

```
HRESULT hr = object->ToneFileCount(short *nFiles);
```

Return Value

nFiles	The tone file count variable. This function will return a -1 if an error occurs. Use the RFGetLastError method to return the generated error code.

Remarks

The ToneFileCount method returns the file count from the last successful call to theListToneFiles Method. Use the ListToneFiles method to return the total number of tone files that are currently stored on an RF device.

ToneFileName Method

This member of RFTone is supported on ALL devices.

The ToneFileName method returns the file name of a saved tone file.

VB

```
pszName = object.ToneFileName (nTone)
```

VC++

```
HRESULT hr = object->ToneFileName(short nTone, Cstring
*pszName);
```

Return Value

pszName	The tone file name variable. If this function returns an empty string an error may have occurred. Use the RFGetLastError to return the generated error code.

Parameters

nTone	The zero based index value tone file whose file name is to be returned. Remember, the first tone in the list is indexed as zero. For example, to return the file name of the third tone file you would pass a 2.

Remarks

For a complete listing of all tone files on an RF device simply loop the ToneFileName method for the entire number of files returned by ToneFileCount.

RFTone Samples

Using both Visual Basic and Visual C++ sample code, the following set of applications will demonstrate the implementation and capabilities of the RFTone object and its methods. Both applications will create a multi-note tone file, save the tone file to the RF device, play the tone file, alter the tone file, then delete the tone from the device when the application is finished.

VB Example

```
' RFTone Object Demo Application
'
' Import a sleep function from the kernel32 dll
Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)
' Declare an RFIO object for use throughout the
' demo application.
Public rfConsole As New RFIO
' Declare an RFMenu object for use throughout the
' demo application.
Public rfTon As New RFTONE
' strBuffer is a temp storage buffer
Dim strBuffer As String
' bDone is a loop control variable
Dim bDone As Boolean
' nTmp is a temporary integer
Dim nTmp As Integer
' nTmp2 is a temporary integer
Dim nTmp2 As Integer

Dim nLcv as Integer


Public Sub Main()
' This application demonstrates the use of the various RFTone
' methods. It assumes a display area of 20x8. In practice, one
' should use an RFTerminal object to format output to fit the
' current device. The following lines clear the display and output
' the app title in reverse video.
    rfConsole.RFPrint 0, 0, "   Welcome To The   ",
              WLCLEAR + WLREVERSE
    rfConsole.RFPrint 0, 1, "    RFTone Demo!    ", WLREVERSE
    rfConsole.RFPrint 0, 3, "Hit a key to proceed", WLNORMAL
' Hide the cursor for a cleaner display
    rfConsole.RFPrint 0, 9, "", WLNORMAL
'
' Flush the output buffer and await a keystroke/scan
```

```
    If rfConsole.GetEvent() = "" Then
        GoTo ExitApp
    End If  ' If rfConsole.GetEvent() = ""

' An RF application can respond to user input in three ways: not at
' all, through the display, or through the speaker. Each has its
' benefits and drawbacks. By using the RF device's speaker, an
' application can quickly respond to user input with a simple
' acknowledgement. Tones can also be used to denote error conditions,
' status reports, and application progress.
' This application demonstrates the methods for creating, storing,
' using, & deleting tones from an RF device.
' Add some notes to the rfTone object
    For nLcv = 0 To 9
        If rfTone.AddTone(440 + (nLcv * 100), 200) = False Then
            GoTo ExitApp
        End If  ' If rfTone.AddTone(440 + (nLcv * 100), 200) = False
    Next nLcv   ' For nLcv = 0 To 9

' Store the tone object on the RF device under "ToneDemo"
    If rfTone.StoreTone("ToneDemo") = False Then
        GoTo ExitApp
    End If  ' If rfTone.StoreTone("ToneDemo") = False
' Play the tone. One might create various tones for data
' validation, etc.
    If rfTone.PlayTone("ToneDemo") = False Then
        If rfTone.RFGetLastError() <> WLNOTINITIALIZED Then
            rfTone.DeleteToneFile "ToneDemo"
        End If  ' If rfTone.RFGetLastError() <> WLNOTINITIALIZED
        GoTo ExitApp
    End If  ' If rfTone.PlayTone("ToneDemo") = False
' To demonstrate some of the other RFTone methods, we will read
' a tone from the RF device,
    rfTone.ClearToness    ' Gives us a fresh object to work with
    If rfTone.GetToneFile("ToneDemo") < 0 Then
        If rfTone.RFGetLastError() <> WLNOTINITIALIZED Then
            rfTone.DeleteToneFile "ToneDemo"
        End If  ' If rfTone.RFGetLastError() <> WLNOTINITIALIZED
        GoTo ExitApp
    End If  ' If rfTone.GetToneFile("ToneDemo") < 0
' double the duration of each individual note,
    nTmp = rfTone.ToneCount() ' Store the number of notes
                              ' in a temp variable
    For nLcv = 0 To nTmp - 1
        nTmp2 = rfTone.GetDuration(nLcv) * 2
```

```
            If nTmp2 < 0 Then
                GoTo ExitApp
            End If  ' If nTmp2 < 0
            If rfTone.SetDuration(nTmp2, nLcv) = False Then
                GoTo ExitApp
            End If  ' If rfTone.SetDuration(nTmp2, nLcv) = False
        Next nLcv

    ' store the new tone on the RF device over the old,
        If rfTone.StoreTone("ToneDemo") = False Then
            If rfTone.RFGetLastError() <> WLNOTINITIALIZED Then
                rfTone.DeleteToneFile "ToneDemo"
            End If  ' If rfTone.RFGetLastError() <> WLNOTINITIALIZED
            GoTo ExitApp
        End If  ' If rfTone.StoreTone("ToneDemo") = False

    ' and play it.
        If rfTone.PlayTone("ToneDemo") = False Then
            If rfTone.RFGetLastError() <> WLNOTINITIALIZED Then
                rfTone.DeleteToneFile "ToneDemo"
            End If  ' If rfTone.RFGetLastError() <> WLNOTINITIALIZED
            GoTo ExitApp
        End If  ' If rfTone.PlayTone("ToneDemo") = False

    ' Anytime an application changes the configuration of an RF device,
    ' it should return it to its original values during cleanup
        rfTone.DeleteToneFile "ToneDemo"

    ExitApp:
    End Sub
```

VC++ Example

```
////////////////////////////////////////////////////////
//      Place RF application's main code here
IRFIO        rfConsole;        // Declare an RFIO interface object
IRFTone      rfTone;           // Declare an RFTone interface object
Cstring      csBuffer;         // Storage buffer for data strings
Short        nTmp;            // Temporary short value

    // Create an RFIO object in the automated server for our use
    if (rfConsole.CreateDispatch ("WAVELINKOLE.RFIO") == FALSE) ↵
        return FALSE;
    // Create an RFTone object in the automated server for our use
    if (rfTone.CreateDispatch ("WAVELINKOLE.RFTONE") == FALSE) ↵
        return FALSE;

    // This application demonstrates the use of the various RFTone
    // methods. It assumes a display area of 20x8. In practice, one
    // should use an RFTerminal object to format output to fit the
    // current device.
    // The following lines clear the display and output the app
    // title in reverse video.
    rfConsole.RFPrint (0, 0, "   Welcome To The   ", ↵
              WLCLEAR | WLREVERSE);
    rfConsole.RFPrint (0, 1, " RFTone Method Demo ", WLREVERSE);
    rfConsole.RFPrint (0, 3, "Hit a key to proceed", WLNORMAL);
    // Hide the cursor for a cleaner display
    rfConsole.RFPrint (0, 9, "", WLNORMAL);

    // Flush the output buffer and await a keystroke/scan
    if ((rfConsole.GetEvent ()).IsEmpty () == TRUE) {
        // Release the RFIO & RFTone objects
        rfConsole.ReleaseDispatch ();
        rfTone.ReleaseDispatch ();
        return FALSE;
    } // if ((rfConsole.GetEvent ()).IsEmpty () == TRUE)

    // An RF application can respond to user input in three ways:
    // not at all, through the display, or through the speaker. Each
    // has its benefits and drawbacks. By using the RF device's
    // speaker, an application can quickly respond to user input
    // with a simple acknowledgement. Tones can also be used to
    // denote error conditions, status reports, and application
    // progress.
    // This application demonstrates the methods for creating,
    // storing, using, & deleting tones from an RF device.
    // Add some notes to the rfTone object
```

```
for (int nLcv = 0; nLcv < 10; ++nLcv) {
    if (rfTone.AddTone (440 + (nLcv * 100), 200) == FALSE) {
        // Release the RFIO & RFTone objects
        rfConsole.ReleaseDispatch ();
        rfTone.ReleaseDispatch ();
        return FALSE;
    } // if (rfTone.AddTone (440 + (nLcv * 100), 200) == FALSE)
} // for (int nLcv = 0; nLcv < 10; ++nLcv)

// Store the tone object on the RF device under "ToneDemo"
if (rfTone.StoreTone ("ToneDemo") == FALSE) {
    // Release the RFIO & RFTone objects
    rfConsole.ReleaseDispatch ();
    rfTone.ReleaseDispatch ();
    return FALSE;
} // if (rfTone.StoreTone ("ToneDemo") == FALSE)

// Play the tone. One might create various tones for data
// validation, etc.
if (rfTone.PlayTone ("ToneDemo") == FALSE) {
    if (rfTone.RFRFGetLastError() != WLNOTINITIALIZED)
        rfTone.DeleteToneFile ("ToneDemo");
    // Release the RFIO & RFTone objects
    rfConsole.ReleaseDispatch ();
    rfTone.ReleaseDispatch ();
    return FALSE;
} // if (rfTone.PlayTone ("ToneDemo") == FALSE)

// To demonstrate some of the other RFTone methods, we will
// read a tone from the RF device
rfTone.ClearToness (); // Gives us a fresh object to work with
if (rfTone.GetToneFile ("ToneDemo") < 0) {
    if (rfTone.RFRFGetLastError() != WLNOTINITIALIZED)
        rfTone.DeleteToneFile ("ToneDemo");
    // Release the RFIO & RFTone objects
    rfConsole.ReleaseDispatch ();
    rfTone.ReleaseDispatch ();
    return FALSE;
} // if (rfTone.GetToneFile ("ToneDemo") < 0)

// double the duration of each individual note,
nTmp = rfTone.ToneCount ();        // Store the number of notes
```

```
                                        // in a temp variable
for (nLcv = 0; nLcv < nTmp; ++nLcv)
    rfTone.SetDuration (rfTone.GetDuration (nLcv) * 2, nLcv);
// store the new tone on the RF device over the old,
if (rfTone.StoreTone ("ToneDemo") == FALSE) {
    if (rfTone.RFRFGetLastError() != WLNOTINITIALIZED)
        rfTone.DeleteToneFile ("ToneDemo");
    // Release the RFIO & RFTone objects
    rfConsole.ReleaseDispatch ();
    rfTone.ReleaseDispatch ();
    return FALSE;
} // if (rfTone.StoreTone ("ToneDemo") == FALSE)

// and play it.
if (rfTone.PlayTone ("ToneDemo") == FALSE) {
    if (rfTone.RFRFGetLastError() != WLNOTINITIALIZED)
        rfTone.DeleteToneFile ("ToneDemo");
    // Release the RFIO & RFTone objects
    rfConsole.ReleaseDispatch ();
    rfTone.ReleaseDispatch ();
    return FALSE;
} // if (rfTone.PlayTone ("ToneDemo") == FALSE)

// Anytime an application changes the configuration of an RF
// device, it should return it to its original values during
// cleanup
rfTone.DeleteToneFile ("ToneDemo");

// Release the RFIO & RFTone objects
rfConsole.ReleaseDispatch ();
rfTone.ReleaseDispatch ();
```

The WaveLink Widget Extension

The WaveLink Widget Extension contains the widget functionality for the WaveLink Development Library. The WaveLink Widget Extension contains over 70 objects, properties, and methods that allow you to create standard buttons, checkboxes, popup triggers, selector triggers, repeater buttons, pushbuttons, hotspots, menubars, bitmaps, and other widgets for your wireless applications.

WaveLinkFactory Object

The WaveLinkFactory object is supported on: Palm, CE

The WaveLinkFactory object contains the methods necessary to build all types of widget objects.

Properties

DefaultCoordinateType
DefaultDisplayBackColor
DefaultDisplayFlags
DefaultDisplayFontSize
DefaultDisplayFontType
DefaultDisplayForeColor

Methods

CreateBitmap
CreateButton
CreateCheckbox
CreateField
CreateHotspot
CreateLabel
CreateMenubar
CreatePopupTrigger
CreatePushButton
CreateRepeaterButton
CreateSelectorTrigger

Prog IDs

"WaveLink.WaveLinkFactory"

Remarks

The WaveLinkFactory object automates the process of creating widgets. Use the WaveLinkWidget object to specify widget characteristics not available in the WaveLinkFactory object.

Note: To cause widgets to display on the RF device, you must use the StoreWidgets method. To return input from a widget, use the RFInput or GetEvent method. These two methods automatically return the widget ID. To process input from the widget based on the widget ID, use the LastExtendedType method. To process input from the widget based on its general input type (scanned, keyed, widget input, etc.) use the LastInputType method.

Property Summary

DefaultCoordinateType Property
- Stores and retrieves the default coordinate type of the WaveLinkFactory object.

DefaultDisplayBackColor Property
- Stores and retrieves the default background color of the WaveLinkFactory object.

DefaultDisplayFlags Property
- Stores and retrieves the default label formatting flags of the WaveLinkFactory object.

DefaultDisplayFontSize Property
- Stores and retrieves the size of the default display font of the WaveLinkFactory object.

DefaultDisplayFontType Property
- Stores and retrieves the name of the default display font of the WaveLinkFactory object.

DefaultDisplayForeColor Property
- Stores and retrieves the default foreground color of the WaveLinkFactory object.

Method Summary

CreateBitmap Method
- Creates an image based on a bitmap file.

CreateButton Method
- Creates a standard button that triggers an action when the user clicks it.

CreateCheckbox Method
- Creates a checkbox that stores an on/off state.

CreateField Method
- Creates an input box.

CreateHotspot Method
- Creates an invisible area that triggers a specific action when the user taps it.

CreateLabel Method
- Creates positioned text in a proportionally-spaced font.

CreateMenubar Method
- Creates a menubar containing a set of menus.

CreatePopupTrigger Method
- Creates a popup trigger that displays a pop-up list when the user taps it.

CreatePushButton Method
- Creates a set of buttons that store an on/off state.

CreateRepeaterButton Method

- Creates a button that implies that something will be scrolled or incremented when the user clicks it.

CreateSelectorTrigger Method

- Creates a button that implies that a dialog box will open when the user clicks it.

DefaultCoordinateType Property

This member of WaveLinkFactory is supported on: Palm, CE

The DefaultCoordinateType property stores and retrieves the default coordinate type of the current WaveLinkFactory object.

VB

```
CoordType = object.DefaultCoordinateType

object.DefaultCoordinateType = CoordType
```

VC++

```
HRESULT hr = object->
get_DefaultCoordinateType(CoordinateTypes *CoordType);

HRESULT hr = object->
put_DefaultCoordinateType(CoordinateTypes CoordType);
```

Return Values

CoordType	The variable that retrieves the default coordinate type (see Remarks)

Parameters

CoordType	The variable that specifies the default coordinate type (see Remarks)

Remarks

The possible values are:

BYCELL	Position the widget using cell coordinates (rows and columns)
BYPIXEL	Position the widget using pixel coordinates
BYPERCENTAGE	Position the widget using percentage coordinates

The DefaultCoordinateType property determines how the application interprets the values entered for the position, width, and height of the widget. The default value is BYCELL.

Example

See the CreateBitmap method

DefaultDisplayBackColor Property

This member of WaveLinkFactory is supported on: CE

The DefaultDisplayBackcolor property stores and retrieves the default background color of the current WaveLinkFactory object.

VB

```
DsplBackColor = object.DefaultDisplayBackColor

object.DefaultDisplayBackColor = DsplBackColor
```

VC++

```
HRESULT hr = object->get_DefaultDisplayBackColor(long
*DsplBackColor);

HRESULT hr = object->put_DefaultDisplayBackColor(long
DsplBackColor);
```

Return Values

BackColor	The variable that retrieves the default background color

Parameters

BackColor	The variable that stores the default background color. Pass a value of –1 to use the default system background color.

Remarks

The background color must be specified in six-byte RGB format (example, FFFFFF).

DefaultDisplayFlags Property

This member of WaveLinkFactory is supported on: Palm, CE

The DefaultDisplayFlags property stores and retrieves the default label formatting flags of the current WaveLinkFactory object.

VB

```
DsplFlags = object.DefaultDisplayFlags

object.DefaultDisplayFlags = DsplFlags
```

VC++

```
HRESULT hr = object->get_DefaultDisplayFlags(long
*DsplFlags);

HRESULT hr = object->put_DefaultDisplayFlags(long
DsplFlags);
```

Return Values

DsplFlags	The variable that retrieves the default label formatting flag (see Remarks)

Parameters

DsplFlags	The variable that specifies the default label formatting flag (see Remarks)

Remarks

Valid settings for the label formatting flag include:

CENTERJUST	- Center-justified label text
LEFTJUST	- Left-justified label text
RIGHTJUST	- Right-justified label text

DefaultDisplayFontSize Property

This member of WaveLinkFactory is supported on: CE

The DefaultDisplayFontSize property stores and retrieves the default font size of the current WaveLinkFactory object.

VB

```
DsplFontSize = object.DefaultDisplayFontSize

object.DefaultDisplayFontSize = DsplFontSize
```

VC++

```
HRESULT hr = object->get_DefaultDisplayFontSize(short
*DsplFontSize);

HRESULT hr = object->put_DefaultDisplayFontSize(short
DsplFontSize);
```

Return Values

DsplFontSize	The variable that retrieves the default font size

Parameters

DsplFontSize	The variable that specifies the default font size. Pass a value of –1 to use the default system font size.

DefaultDisplayFontType Property

This member of WaveLinkFactory is supported on: CE

The DefaultDisplayFontType property stores and retrieves the default font of the current WaveLinkFactory object.

VB

```
DsplFontName = object.DefaultDisplayFontType

object.DefaultDisplayFontType = DsplFontName
```

VC++

```
HRESULT hr = object->get_DefaultDisplayFontType(BSTR
*DsplFontName);

HRESULT hr = object->put_DefaultDisplayFontType(BSTR
DsplFontName);
```

Return Values

DsplFontName	The variable that retrieves the default font

Parameters

DsplFontName	The variable that specifies the default font. Pass a value of null to use the default system font.

DefaultDisplayForeColor Property

This member of WaveLinkFactory is supported on: CE

The DefaultDisplayForeColor property stores and retrieves the default foreground color of the current WaveLinkFactory object.

VB

```
DsplForeColor = object.DefaultDisplayForeColor

object.DefaultDisplayForeColor = DsplForeColor
```

VC++

```
HRESULT hr = object->get_DefaultDisplayForeColor(long *
DsplForeColor);

HRESULT hr = object->put_DefaultDisplayForeColor(long
DsplForeColor);
```

Return Values

DsplForeColor	The variable that retrieves the default foreground color

Parameters

DsplForeColor	The variable that specifies the default foreground color. Pass a value of –1 to use default system foreground color.

Remarks

The color must be specified in six-byte RGB format (example, FFFFFF).

CreateBitmap Method

This member of WaveLinkFactory is supported on: Palm, CE

The CreateBitmap method creates an image based on a bitmap file.

VB

```
object.CreateBitmap XCoord, YCoord, Width, Height,
BitmapFile, MyColl
```

VC++

```
HRESULT hr = object->CreateBitmap(short XCoord, short
YCoord, short Width, short Height, BSTR BitmapFile,
WaveLinkWidgetCollection MyColl);
```

Parameters

XCoord	The starting left coordinate of the widget (see Remarks)
YCoord	The starting top coordinate of the widget (see Remarks)
Width	The horizontal extent of the widget. Pass a value of 0 to set the width automatically.
Height	The vertical extent of the widget Pass a value of 0 to set the height automatically.
BitmapFile	The file name of the bitmapped image. The file name must include the full path.
MyColl	The name of the collection that will contain the widget

Remarks

The DefaultCoordinateType property determines how the application interprets the values entered for the position, width, and height of the widget.

You can set the size of the widget to AUTOSIZE by setting the width and height to zero (0).

The CreateBitmap method automatically stores the path for the bitmap image in the SpecialString property.

Example

```
' VB Sample Code
Dim fileIface As New RFFILE
Dim ioIface As New RFIO
Dim myCollection As New WaveLinkWidgetCollection
Dim myFactory As New WaveLinkFactory
.
.
.
  ' The following code illustrates an easy way to transfer the
  ' image file
  fileIface.RFTransferFile "C:\temp\logo.bmp", "logo.bmp", True

  ' you may clear the screen
  ioIface.RFPrint 0, 35, "", WLCLEAR

  myFactory.DefaultCoordinateType = BYPIXEL
  myFactory.CreateBitmap 20, 20, 35, 25, "logo.bmp", myCollection

  myCollection.StoreWidgets
```

CreateButton Method

This member of WaveLinkFactory is supported on: Palm, CE

The CreateButton method creates a standard labeled button that triggers a specific action when the user "clicks" it.

VB

```
object.CreateButton XCoord, YCoord, Width, Height,
LabelText, MyColl
```

VC++

```
HRESULT hr = object->CreateButton(short XCoord, short
YCoord, short Width, short Height, BSTR LabelText,
WaveLinkWidgetCollection MyColl);
```

Parameters

XCoord	The starting left coordinate of the widget (see Remarks)
YCoord	The starting top coordinate of the widget (see Remarks)
Width	The horizontal extent of the widget. Pass a value of 0 to set the width automatically.
Height	The vertical extent of the widget. Pass a value of 0 to set the height automatically.
LabelText	The display text of the widget
MyColl	The name of the collection that will contain the widget

Remarks

The default value returned by a button widget through an RFInput or GetEvent call will be the display text.

The DefaultCoordinateType property determines how the application interprets the values entered for the position, width, and height of the widget.

You can set the size of the widget to AUTOSIZE by setting the width and height to zero (0). This setting sizes the button widget according to the size of the text that it contains.

Example

```
' VB Sample Code
Dim wlfactory As New WaveLinkFactory
Dim wlwidgcoll As New WaveLinkWidgetCollection
Dim wlio As New RFIO
Dim wlterm As New RFTerminal
Dim myWidget As WaveLinkWidget
.
.
.
  ' In this example, give the widget some context
  wlio.RFPrint 0, 0, "WaveLink Corporation", WLCLEAR + WLREVERSE
  wlio.RFPrint 0, 1, "   Auto Detailing   ", WLNORMAL
  ' create the widget
  Set myWidget = wlfactory.CreateButton(4, _
                 (wlterm.TerminalHeight - 2), _
                 0, 0, "  OK  ", wlwidgcoll)

  wlwidgcoll.StoreWidgets
```

CreateCheckbox Method

This member of WaveLinkFactory is supported on: Palm, CE

The CreateCheckbox method creates a widget that stores an on/off state. The on/off state switches when the user taps it.

VB

```
object.CreateCheckbox XCoord, YCoord, Width, Height,
LabelText, InitialState, MyColl
```

VC++

```
HRESULT hr = object->CreateCheckbox(short XCoord, short
YCoord, short Width, short Height, BSTR LabelText,
CheckboxState InitialState, WaveLinkWidgetCollection
MyColl);
```

Parameters

XCoord	The starting left coordinate of the widget (see Remarks)
YCoord	The starting top coordinate of the widget (see Remarks)
Width	The horizontal extent of the widget. Pass a value of 0 to set the width automatically.
Height	The vertical extent of the widget. Pass a value of 0 to set the height automatically.
LabelText	The display text of the widget
InitialState	The initial state of the checkbox (CHECKED, UNCHECKED, UNDETERMINED)
MyColl	The name of the collection that will contain the widget

Remarks

The default value returned by a checkbox widget through an RFInput or GetEvent call will be the checkbox state. The possible values for the checkbox state are:

0 - UNCHECKED	This constant represents an unchecked checkbox state
1 - CHECKED	This constant represents a checked checkbox state

2 - UNDETERMINED	This constant represents an undetermined checkbox state

The DefaultCoordinateType property determines how the application interprets the values entered for the position, width, and height of the widget.

You can set the size of the widget to AUTOSIZE by setting the width and height to zero (0). It is recommended that you use the AUTOSIZE setting for the height and width of the checkbox. Using a value other than AUTOSIZE will not affect the size of the widget, only the extent of the clickable area surrounding it.

Example

```
' VB Sample Code
Dim wlio As New RFIO
Dim wlfactory As New WaveLinkFactory
Dim wlcheckboxcoll As New WaveLinkWidgetCollection
Dim PassCarCheckBox As WaveLinkWidget
.
.
.
  ' In this example, give the widget a context
  wlio.RFPrint 0, 0, "WaveLink Corporation", WLCLEAR + WLREVERSE
  wlio.RFPrint 0, 1, "   Auto Detailing   ", WLNORMAL
  wlio.RFPrint 0, 3, "   Select Vehicle   ", _
       WLNORMAL + WLFLUSHOUTPUT
  ' Create the widget
  Set PassCarCheckBox = wlfactory.CreateCheckbox(0, 5, 0, 0, _
                        "Passenger Auto", UNCHECKED, _
                        wlcheckboxcoll)

  wlcheckboxcoll.StoreWidgets
```

CreateField Method

This member of WaveLinkFactory is supported on: Palm, CE

The CreateField method creates an input box.

VB

```
object.CreateField XCoord, YCoord, Width, Height, FieldText,
MyColl
```

VC++

```
HRESULT hr = object->CreateField(short XCoord, short YCoord,
short Width, short Height, BSTR FieldText,
WaveLinkWidgetCollection MyColl);
```

Parameters

XCoord	The starting left coordinate of the widget (see Remarks)
YCoord	The starting top coordinate of the widget (see Remarks)
Width	The horizontal extent of the widget. Pass a value of 0 to set the width automatically.
Height	The vertical extent of the widget. Pass a value of 0 to set the height automatically.
FieldText	The default text contained in the field
MyColl	The name of the collection that will contain the widget

Remarks

The DefaultCoordinateType property determines how the application interprets the values entered for the position, width, and height of the widget.

You can set the size of the widget to AUTOSIZE by setting the width and height to zero (0).

Example

```
' VB Sample Code
Dim iolIface As New RFIO
Dim myCollection As New WaveLinkWidgetCollection
Dim myFactory As New WaveLinkFactory
Dim myWidget As WaveLinkWidget
.
.
.
    ' Create and title a field to get the user's name
    ioIface.RFPrint 3, 2, "Enter your Name:", WLCLEAR
    Set myWidget = myFactory.CreateField(3, 3, 12, _
                   1, "your name", myCollection)

    mycollection.StoreWidgets
```

CreateHotspot Method

This member of WaveLinkFactory is supported on: Palm, CE

The CreateHotspot method creates an invisible area that triggers a specific action when the user taps it.

VB

```
object.CreateHotspot XCoord, YCoord, Width, Height, MyColl
```

VC++

```
HRESULT hr = object->CreateHotspot(short XCoord, short
YCoord, short Width, short Height, WaveLinkWidgetCollection
MyColl);
```

Parameters

XCoord	The starting left coordinate of the widget (see Remarks)
YCoord	The starting top coordinate of the widget (see Remarks)
Width	The horizontal extent of the widget. This value must be greater than zero (0).
Height	The vertical extent of the widget. This value must be greater than zero (0).
MyColl	The name of the collection that will contain the widget

Remarks

The DefaultCoordinateType property determines how the application interprets the values entered for the position, width, and height of the widget.

Example

```
' VB Sample Code
Dim ioIface As New RFIO
Dim myCollection As New WaveLinkWidgetCollection
Dim myFactory As New WaveLinkFactory
Dim my Hotspot As WaveLinkWidget
.
.
.
  ' In this example, clear the screen and hide the cursor
  ioIface.RFPrint 0, 25, "", WLCLEAR
  myFactory.DefaultCoordinateType = BYPIXEL
  ' Give some context for the hotspot
  myFactory.CreateBitmap 20, 20, 35, 25, _
            "logo.bmp", myCollection
  ' Create the widget
  Set myHotspot = myFactory.CreateHotspot 20, 20, 35, 25, _
                  myCollection

  myCollection.StoreWidgets
```

CreateLabel Method

This member of WaveLinkFactory is supported on: Palm, CE

The CreateLabel method creates positioned text that is in a porportionally spaced font and has the same "lifetime" as the other widgets.

VB

```
object.CreateLabel XCoord, YCoord, Width, Height, LabelText,
MyColl
```

VC++

```
HRESULT hr = object->CreateLabel(short XCoord, short YCoord,
short Width, short Height, BSTR LabelText,
WaveLinkWidgetCollection MyColl);
```

Parameters

XCoord	The starting left coordinate of the widget (see Remarks)
YCoord	The starting top coordinate of the widget (see Remarks)
Width	The horizontal extent of the widget. Pass a value of 0 to set the width automatically.
Height	The vertical extent of the widget. Pass a value of 0 to set the height automatically.
LabelText	The display text of the widget
MyColl	The name of the collection that will contain the widget

Remarks

The DefaultCoordinateType property determines how the application interprets the values entered for the position, width, and height of the widget.

You can set the size of the widget to AUTOSIZE by setting the width and height to zero (0). It is recommended that you use the AUTOSIZE setting for the height and width of the label. If you set these values too small, the display text will be cropped.

Example

```
' VB Sample Code
Dim myCollection As New WaveLinkWidgetCollection
Dim myFactory As New WaveLinkFactory
.
.
.
    myFactory.CreateLabel 10, 10, 0, 0, _
              "Click to Enter", myCollection

    myCollection.StoreWidgets
```

CreateMenubar Method

This member of WaveLinkFactory is supported on: Palm, CE

The CreateMenubar method creates a menubar containing a set of menus. Each menu is associated with a pop-up list that activates when the user taps it.

VB

```
object.CreateMenubar WaveLinkMenubarInfo MenuBarInfo,
WaveLinkWidgetCollection MyColl
```

VC++

```
HRESULT hr = object->CreateMenubar(WaveLinkMenubarInfo
MenuBarInfo, WaveLinkWidgetCollection MyColl);
```

Parameters

MenuBarInfo	The name of the WaveLinkMenubarInfo object to associate with the widget
MyColl	The name of the collection that will contain the widget

Remarks

The menubar displays menu names based on stored RFMenu configurations. The menu configurations used by the menubar must be stored on the device before the menubar can access them. The menubar widget will return the menu index of the user-selected option.

Use the WaveLinkMenubarInfo object to create a list of menus for the menubar widget. When creating the widget object, the CreateMenubar method automatically calls the SetSpecialInfo method which assigns the formatted list of menus to the SpecialString property. If you want to modify the list of menus for the current menubar widget, use the SpecialString property.

Example

```
' VB Sample Code
Dim myMenInfo As New WaveLinkMenubarInfo
Dim mnuIface As New RFMenu
Dim myCollection As New WaveLinkWidgetCollection
Dim myFactory As New WaveLinkFactory
Dim myWidget As WaveLinkWidget
.
.
.
    'Create the menus that will appear in the main menu bar
    mnuIface.ResetMenu
    mnuIface.AddTitleLine "Widget Demo's"
    mnuIface.AddOption "Buttons"
    mnuIface.AddOption "Pen Capture"
    mnuIface.StoreMenu "MenOne"

    mnuIface.ResetMenu
    mnuIface.AddTitleLine "Exit"
    mnuIface.AddOption "Quit"
    mnuIface.StoreMenu "MenTwo"

    'Insert the two menus into the WaveLinkMenuBarInfo object,
    'using a -1 as the index automatically
    'tacks the menu to the end of the list
    myMenInfo.Insert -1, "MenOne"
    myMenInfo.Insert -1, "MenTwo"
.
.
.
    'Create the menubar
    Set myWidget = myFactory.CreateMenubar myMenInfo, myCollection

    myCollection.StoreWidgets
```

CreatePopupTrigger Method

This member of WaveLinkFactory is supported on: Palm, CE

The CreatePopupTrigger method creates a popup trigger that displays a pop-up list when the user taps it. The currently selected item appears to the right of the trigger.

VB

```
object.CreatePopupTrigger XCoord, YCoord, Width, Height,
MenuName, InitialOption, MyColl
```

VC++

```
HRESULT hr = object->CreatePopupTrigger(short XCoord, short
YCoord, short Width, short Height, BSTR MenuName, short
InitialOption, WaveLinkWidgetCollection MyColl);
```

Parameters

XCoord	The starting left coordinate of the widget (see Remarks)
YCoord	The starting top coordinate of the widget (see Remarks)
Width	The horizontal extent of the widget. Pass a value of 0 to set the width automatically.
Height	The vertical extent of the widget. Pass a value of 0 to set the height automatically.
MenuName	The name of the RFMenu configuration associated with the widget
InitialOption	The initial menu-index value of the widget (for example, passing a 1 will indicate the first menu entry in the RFMenu configuration)
MyColl	The name of the collection that will contain the widget

Remarks

The popup trigger populates its menu options with a stored RFMenu configuration. The menu must be stored on the device before the popup trigger can access it. The widget will return the menu index of the user-selected option.

The CreatePopupTrigger method automatically stores the menu name associated with the current widget in the SpecialString property of the WaveLinkWidget object.

The DefaultCoordinateType property determines how the application interprets the values entered for the position, width, and height of the widget. You can set the size of the widget to AUTOSIZE by setting the width and height to zero (0). It is recommended that you use the AUTOSIZE setting for the height and width of the popup trigger. Using a value other than AUTOSIZE will not affect the size of the widget, only the extent of the clickable area surrounding it.

Example

```
' VB Sample Code
Dim wlfactory As New WaveLinkFactory
Dim wlmaincoll As New WaveLinkWidgetCollection
Dim WelcomeMenu As WaveLinkWidget
Dim wlmenu As New RFMenu
.
.
.
    wlmenu.ResetMenu
    wlmenu.SetMenuWidth 18
    wlmenu.AddOption "Basic Car Wash"
    wlmenu.AddOption "Basic Wash/Wax"
    wlmenu.AddOption "Carnuba Wax"
    wlmenu.AddOption "Interior Clean"
    wlmenu.AddOption "Full Detail"
    wlmenu.AddOption "Special Detail"
    wlmenu.AddOption "Device Version"
    wlmenu.AddOption "Exit"
    wlmenu.StoreMenu "MainMenu"
.
.
.
    Set WelcomeMenu = wlfactory.CreatePopupTrigger(3, 5, 0, 0, _
                    "MainMenu", 1, wlmaincoll)

    wlmaincoll.StoreWidgets
```

CreatePushButton Method

This member of WaveLinkFactory is supported on: Palm, CE

The CreatePushButton method creates a set of buttons that store an on/off state. When the user selects a button, the selected button stores the "on" state, and all other buttons store the "off" state. Only one button can store the "on" state at any time.

VB

```
object.CreatePushButton XCoord, YCoord, Width, Height,
MenuName, InitialOption, MyColl
```

VC++

```
HRESULT hr = object->CreatePushButton(short XCoord, short
YCoord, short Width, short Height, BSTR MenuName, short
InitialOption, WaveLinkWidgetCollection MyColl);
```

Parameters

XCoord	The starting left coordinate of the widget (see Remarks)
YCoord	The starting top coordinate of the widget (see Remarks)
Width	The horizontal extent of the widget. Pass a value of 0 to set the width automatically.
Height	The vertical extent of the widget. Pass a value of 0 to set the height automatically.
MenuName	The name of the RFMenu configuration associated with the widget.
InitialFlag	The initial state of the widget (see Remarks).
MyColl	The name of the collection that will contain the widget

Remarks

The push button widget populates its menu options with a stored RFMenu configuration. The menu must be stored on the device before the push button can access it. The widget will return the menu index of the user-selected option.

The CreatePushButton method automatically stores the menu name associated with the current widget in the SpecialString property of the WaveLinkWidget object.

The possible values for the *InitialFlag* parameter are:

INITSTANDARD	Initial state of the widget is standard (shown and enabled)
INITHIDDEN	Initial state of the widget is hidden
INITDISABLED	Initial state of the widget is disabled

A disabled widget will appear on the RF screen, but it will not return when you click on it. The appearance of a disabled widget is platform dependent. Typically, a disabled widget will either be grayed out or a crosshatch pattern will fill its display text.

You can set other properties specific to the push button widget using the DisplayFlags property of the WaveLinkWidget object.

The DefaultCoordinateType property determines how the application interprets the values entered for the position, width, and height of the widget. You can set the size of the widget to AUTOSIZE by setting the width and height to zero (0). It is recommended that you use the AUTOSIZE setting for the height and width of the push button. Changing this value from its default will not affect the size of the widget, only the extent of the clickable area surrounding it.

Example

```
' VB Sample Code
Dim wlfactory As New WaveLinkFactory
Dim wlmaincoll As New WaveLinkWidgetCollection
Dim wlio As New RFIO
Dim WelcomeMenu As WaveLinkWidget
Dim wlmenu As New RFMenu
.
.
.
  wlmenu.ResetMenu
  wlmenu.SetMenuWidth 18
  wlmenu.AddOption "1"
  wlmenu.AddOption "2"
  wlmenu.AddOption "3"
  wlmenu.StoreMenu "DegofDif"

  ' In this example, show some context for the widget
  wlio.RFPrint 0, 2, "Degree of   ", WLNORMAL
  wlio.RFPrint 0, 3, "Difficulty: ", WLNORMAL
  ' Create the widget
  Set WelcomeMenu = wlfactory.CreatePushButton(3, 5, 0, 0, _
                    "DegofDif", 1, wlmaincoll)

  WelcomeMenu.DisplayFlags = PBFIXED
  wlmaincoll.StoreWidgets
```

CreateRepeaterButton Method

This member of WaveLinkFactory is supported on: Palm, CE

The CreateRepeaterButton method creates a button that implies that something will be scrolled or incremented. The four available buttons are: left arrow, right arrow, up arrow, and down arrow.

VB

```
object.CreateRepeaterButton XCoord, YCoord, Width, Height,
Type, MyColl
```

VC++

```
HRESULT hr = object->CreateRepeaterButton(short XCoord,
short YCoord, short Width, short Height, RepeaterType Type,
WaveLinkWidgetCollection MyColl);
```

Parameters

XCoord	The starting left coordinate of the widget (see Remarks)
YCoord	The starting top coordinate of the widget (see Remarks)
Width	The horizontal extent of the widget. Pass a value of 0 to set the width automatically.
Height	The vertical extent of the widget. Pass a value of 0 to set the height automatically.
Type	The type of repeater button(LEFTARROW, RIGHTARROW, UPARROW, DOWNARROW)
MyColl	The name of the collection that will contain the widget

Remarks

The possible repeater button types are:

LEFTARROW

RIGHTARROW

UPARROW

DOWNARROW

The DefaultCoordinateType property determines how the application interprets the values entered for the position, width, and height of the widget.

You can set the size of the widget to AUTOSIZE by setting the width and height to zero (0). It is recommended that you use the AUTOSIZE setting for

the height and width of the repeater button. Changing this value from its default will not affect the size of the widget, only the extent of the clickable area surrounding it.

Example

```
' VB Sample Code
Dim myFactory As New WaveLinkFactory
Dim myCollection As New WaveLinkWidgetCollection
Dim myLeftWidget As WaveLinkWidget
Dim myRightWidget As WaveLinkWidget
.
.
.
    Set myLeftWidget = myFactory.CreateRepeaterButton(1, 7, 0, _
                    0, LEFTARROW, myCollection)
    Set myRightWidget = myFactory.CreateRepeaterButton(19, 7, 0, _
                    0, RIGHTARROW, myCollection)

    myCollection.StoreWidgets
```

CreateSelectorTrigger Method

This member of WaveLinkFactory is supported on: Palm, CE

The CreateSelectorTrigger method creates a button that implies that a new screen will appear when the user "clicks" it.

VB

```
object.CreateSelectorTrigger XCoord, YCoord, Width, Height,
LabelText, MyColl
```

VC++

```
HRESULT hr = object->CreateSelectorTrigger(short XCoord,
short YCoord, short Width, short Height, BSTR LabelText,
WaveLinkWidgetCollection MyColl);
```

Parameters

XCoord	The starting left coordinate of the widget (see Remarks)
YCoord	The starting top coordinate of the widget (see Remarks)
Width	The horizontal extent of the widget. Pass a value of 0 to set the width automatically.
Height	The vertical extent of the widget. Pass a value of 0 to set the height automatically.
LabelText	The display text of the widget
MyColl	The name of the collection that will contain the widget

Remarks

The DefaultCoordinateType property determines how the application interprets the values entered for the position, width, and height of the widget.

You can set the size of the widget to AUTOSIZE by setting the width and height to zero (0). This setting sizes the selector trigger widget according to the size of the text that it contains.

Example

```
' VB Sample Code
Dim wlio As New RFIO
Dim wlfactory As New WaveLinkFactory
Dim wlwidgcoll As New WaveLinkWidgetCollection
Dim wlterm As New RFTerminal
Dim ConfigWidget As WaveLinkWidget
.
.
.
  wlio.RFPrint 0, 0, "WaveLink Corporation", WLCLEAR + WLREVERSE
  wlio.RFPrint 0, 1, "     Test Demo      ", WLNORMAL
  Set ConfigWidget = wlfactory.CreateSelectorTrigger(4, _
                   (wlterm.TerminalHeight - 4), _
                   0, 0, " Configure ", wlwidgcoll)

  wlwidgcoll.StoreWidgets
```

WaveLinkFactory Samples

Using both Visual Basic sample code, the following set of applications will demonstrate the use of the WaveLinkFactory object and its methods.

Visual Basic Sample:

```
Attribute VB_Name = "Module1"
Option Explicit

' Import a sleep function from the kernel32 dll

Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)

Public wlio As New RFIO
Public wlerror As New RFERROR
Public wlmenu As New RFMENU
Public wlbar As New RFBARCODE
Public wlterm As New RFTERMINAL
Public wlfile As New RFFILE

Public wlwidgcoll As New WaveLinkWidgetCollection
Public wlvervidgcoll As New WaveLinkWidgetCollection
Public wlmaincoll As New WaveLinkWidgetCollection
Public wlcheckboxcoll As New WaveLinkWidgetCollection
Public wlyesnocoll As New WaveLinkWidgetCollection
Public wlfactory As New WaveLinkFactory
Public StartWidget As WaveLinkWidget
Public InfoWidget As WaveLinkWidget
Public DoneWidget As WaveLinkWidget
Public DoneVerWidget As WaveLinkWidget
Public DoneOptButtonWidget As WaveLinkWidget
Public WelcomeMenu As WaveLinkWidget
Public PassCarCheckBox As WaveLinkWidget
Public SUVCheckBox As WaveLinkWidget
Public TruckCheckBox As WaveLinkWidget
Public BoatCheckBox As WaveLinkWidget
Public GVBackButton As WaveLinkWidget
Public GVDoneButton As WaveLinkWidget
Public GVHelpButton As WaveLinkWidget
Public VDYesButton As WaveLinkWidget
Public VDNoButton As WaveLinkWidget

Public wlscribble As New WaveLinkScribblePad

'  Global Variables
```

```
Dim bDone As Boolean
Dim bStatus As Boolean
Dim bFirstVersion As Boolean
Dim bExitGetVehicleType As Boolean

Dim nError As Integer
Dim nRslt As Integer
Dim nRFTerminalType As Integer
Dim pszWorkType As String
Dim pszModelType As String

Const WLWIDGETTYPE = 8
Const WLPOPUPTRIGGERTYPE = 7
Const WLMENUBARTYPE = 6
Const PDT1740 = 11
Const PDT2740 = 27
Const WIDGETDEVICE = 1111



Public Sub Main()

  Initialize
  Do Until bDone
    Process
  Loop
  Finish

End Sub

Public Sub Initialize()

  bDone = False
  ' Saves current Barcode Configuration
    bStatus = wlbar.PushBarcode("Defltbar")
    wlbar.ClearBarcodes
    wlbar.StoreBarcode "BarCode", BCENABLED

    nRFTerminalType = wlterm.RawTerminalType
    Select Case nRFTerminalType
      Case PDT1740
        nRFTerminalType = WIDGETDEVICE
      Case PDT2740
        nRFTerminalType = WIDGETDEVICE
    End Select
```

```
  If Welcome() Then
    wlio.SetFillChar "_"
    If bDone = False Then
      wlmenu.ResetMenu
      wlmenu.SetMenuWidth 18
      wlmenu.AddOption "Basic Car Wash"
      wlmenu.AddOption "Basic Wash/Wax"
      wlmenu.AddOption "Carnuba Wax"
      wlmenu.AddOption "Interior Clean"
      wlmenu.AddOption "Full Detail"
      wlmenu.AddOption "Special Detail"
      wlmenu.AddOption "Device Version"
      wlmenu.AddOption "Exit"
      wlmenu.StoreMenu "MainMenu"
      Select Case nRFTerminalType
        Case WIDGETDEVICE
          wlmenu.SetStartColumn 2
          wlmenu.SetStartRow 4
        Case Else
          wlmenu.SetStartColumn 1
          wlmenu.SetStartRow 3
          wlmenu.SetMenuHeight 12
          wlmenu.AddTitleLine "Select Option"
          wlmenu.AddTitleLine " "
      End Select
      wlmenu.StoreMenu "MainMenu"
    End If

    bFirstVersion = True

  End If

End Sub

Public Function Welcome() As Boolean
  Dim pszWelcomeIn As String
  Dim pszWelcomeInDefault As String
  Dim nLastInputType As Integer
  pszWelcomeIn = ""
  pszWelcomeInDefault = ""
  Welcome = True
  wlio.SetFillChar ""
  wlio.RFPrint 0, 0, "WaveLink Corporation", WLCLEAR + WLREVERSE
  wlio.RFPrint 0, 1, "   Auto Detailing   ", WLNORMAL
  Select Case nRFTerminalType
    Case WIDGETDEVICE
```

```
      '  Widget Buttons for Welcome Screen (Start,Done)
      Set StartWidget = wlfactory.CreateButton(4, _
                        (wlterm.TerminalHeight - 2), _
                        0, 0, " Start ", wlwidgcoll)
      Set DoneWidget = wlfactory.CreateButton(11, _
                        (wlterm.TerminalHeight - 2), _
                        0, 0, " Done  ", wlwidgcoll)
      wlwidgcoll.StoreWidgets
    Case Else
      wlio.RFPrint 4, 3, "CtlX to Exit", WLNORMAL
      wlio.RFPrint 0, (wlterm.TerminalHeight - 1), _
           "    Press any Key    ", WLREVERSE
  End Select

  pszWelcomeIn = wlio.RFInput(pszWelcomeInDefault, 1, 1, _
                 33, "BarCode", WLCAPSLOCK, WLNO_RETURN_BKSP)
  nError = wlio.RFGetLastError
  If nError <> WLNOERROR Then
    bDone = True
    Welcome = False
  Else
    nLastInputType = wlio.LastInputType
    Select Case nLastInputType
      Case WLCOMMANDTYPE
        If pszWelcomeIn = Chr$(24) Then
            bDone = True
            Welcome = False
        End If
      Case WLKEYTYPE

      Case WLSCANTYPE

      Case WLWIDGETTYPE
        If wlio.LastExtendedType() = DoneWidget.WidgetID Then
          bDone = True
          Welcome = False
        End If
    End Select
  End If
End Function



Public Sub Process()

  Dim bMenuSelectDone As Boolean
```

```
bMenuSelectDone = False
Do Until bMenuSelectDone = True

  bMenuSelectDone = True
  wlio.RFPrint 0, 0, "WaveLink Corporation", WLCLEAR + WLREVERSE
  wlio.RFPrint 0, 1, "   Auto Detailing   ", _
       WLNORMAL + WLFLUSHOUTPUT
  Select Case nRFTerminalType
    Case WIDGETDEVICE
      wlwidgcoll.ShowWidgets False  'Turn off Welcome Widget
      GetOption
    Case Else
      nRslt = wlmenu.DoMenu("MainMenu")
  End Select

  Select Case nRslt
    Case -2       ' an error has occured
      bStatus = wlerror.ClearError()
      bStatus = wlerror.SetErrorLine("Menu Not Found", 0)
      bStatus = wlerror.Display(0)
      bDone = True
    Case -1       ' Clr or Ctrl X user exit
      bDone = True
    Case Else     ' valid menu selection
      Select Case nRslt
        Case 1
          pszWorkType = "Basic Car Wash"
        Case 2
          pszWorkType = "Basic Wash/Wax"
        Case 3
          pszWorkType = "Carnuba Wax   "
        Case 4
          pszWorkType = "Interior Clean"
        Case 5
          pszWorkType = "Full Detail   "
        Case 6
          pszWorkType = "Special Detail"
        Case 7      ' RF Device Information
          Version
          bMenuSelectDone = False
        Case 8      ' Exit the Application
          bDone = True
        Case Else   ' should not get here
          bStatus = wlerror.ClearError()
          bStatus = wlerror.SetErrorLine("Invalid Option", 0)
          bStatus = wlerror.Display(0)
```

```
        End Select
    End Select
  Loop
  If bDone <> True Then
    GetVehicleType
  End If
End Sub


Public Sub Finish()

' Delete Barcode Configurations from Device
  bStatus = wlfile.RFDeleteFile("BarCode.bar")
' Restore Default Barcode Configuration and Delete from Device
  bStatus = wlbar.PullBarcode("Defltbar.bar")
End Sub

Public Sub GetOption()

  Dim pszOptionIn As String
  Dim pszOptionInDefault As String
  Dim nLastInputType As Integer
  pszOptionInDefault = ""
  Set WelcomeMenu = wlfactory.CreatePopupTrigger(3, 5, 0, 0, _
                    "mainmenu", 1, wlmaincoll)
  Set DoneOptButtonWidget = wlfactory.CreateButton(8, _
                            (wlterm.TerminalHeight - 2), 0, _
                            0, " Done  ", wlmaincoll)

  wlmaincoll.StoreWidgets

  wlio.RFPrint 0, 3, "Select Option: ", WLNORMAL + WLFLUSHOUTPUT
  wlmaincoll.ShowWidgets True
  wlvervidgcoll.ShowWidgets True  ' Done Button
  pszOptionIn = wlio.RFInput(pszOptionInDefault, 1, 1, 33, _
                "BarCode", WLCAPSLOCK, WLNO_RETURN_BKSP)
  If nError <> WLNOERROR Then
    bDone = True
  Else
    nLastInputType = wlio.LastInputType
    Select Case nLastInputType
      Case WLCOMMANDTYPE
        Select Case pszOptionIn
          Case Chr$(27) ' CLR Key
            nRslt = -1
          Case Chr$(24) ' CRTL X Key
```

```
            nRslt = -1
          Case Else
      End Select
      Case WLPOPUPTRIGGERTYPE
        nRslt = Val(pszOptionIn)
      Case WLWIDGETTYPE
        If wlio.LastExtendedType() = DoneOptButtonWidget.WidgetID _
          Then
          nRslt = 8
        End If
      Case Else
    End Select
  End If

  wlmaincoll.ShowWidgets False
End Sub

Public Sub GetVehicleType()

  Dim VTColumn As Integer
  Dim VTRow As Integer
  Dim nLastInputType As Integer
  Dim pszVehicleTypeIn As String
  Dim pszVehicleTypeDefault As String
  bExitGetVehicleType = False
  pszVehicleTypeIn = ""
  pszVehicleTypeDefault = ""
  wlio.RFPrint 0, 0, "WaveLink Corporation", WLCLEAR + WLREVERSE
  wlio.RFPrint 0, 1, "   Auto Detailing   ", WLNORMAL
  wlio.RFPrint 0, 3, "   Select Vehicle   ", _
       WLNORMAL + WLFLUSHOUTPUT
  Do While pszVehicleTypeIn = ""

    Select Case nRFTerminalType
      Case WIDGETDEVICE
        VTColumn = 0    'Hide Cursor
        VTRow = 33      'Hide Cursor
        Set PassCarCheckBox = wlfactory.CreateCheckbox(0, 5, 0, 0, _
                              "Passenger Auto", Unchecked, _
                              wlcheckboxcoll)
        Set SUVCheckBox = wlfactory.CreateCheckbox(0, 6, 0, 0, _
                          "Sport Utility ", Unchecked, _
                          wlcheckboxcoll)
        Set TruckCheckBox = wlfactory.CreateCheckbox(0, 7, 0, 0, _
                            "Pickup Truck  ", Unchecked, _
                            wlcheckboxcoll)
```

```
    Set BoatCheckBox = wlfactory.CreateCheckbox(0, 8, 0, 0, _
                       "Pleasure Boat  ", Unchecked, _
                       wlcheckboxcoll)
    Set GVBackButton = wlfactory.CreateButton(2, _
                       (wlterm.TerminalHeight - 2), 0, 0, _
                       "Previous", wlcheckboxcoll)
    Set GVDoneButton = wlfactory.CreateButton(8, _
                       (wlterm.TerminalHeight - 2), 0, 0, _
                       "   Done    ", wlcheckboxcoll)
    Set GVHelpButton = wlfactory.CreateButton(14, _
                       (wlterm.TerminalHeight - 2), 0, 0, _
                       "   Help    ", wlcheckboxcoll)
    wlcheckboxcoll.StoreWidgets
  Case Else
    wlio.RFPrint 0, 5, "1. Passenger Auto  ", WLNORMAL
    wlio.RFPrint 0, 6, "2. Sport Utility   ", WLNORMAL
    wlio.RFPrint 0, 7, "3. Pickup Truck    ", WLNORMAL
    wlio.RFPrint 0, 8, "4. Pleasure Boat   ", WLNORMAL
    wlio.RFPrint 0, 9, "5. Previous Screen  ", WLNORMAL
    wlio.RFPrint 0, 10, "6. Exit Application ", WLNORMAL
    wlio.RFPrint 0, 12, "Enter Choice ", WLNORMAL
    wlio.RFPrint 0, (wlterm.TerminalHeight - 1), _
         "CtrlX-Exit | F1-Help ", WLREVERSE
    VTColumn = 13   'Show Cursor
    VTRow = 12      'Show Cursor
End Select
pszVehicleTypeIn = wlio.RFInput(pszVehicleTypeDefault, 1, _
                   VTColumn, VTRow, "BarCode", WLCAPSLOCK, _
                   WLNO_RETURN_BKSP)
nError = wlio.RFGetLastError
If nError <> WLNOERROR Then
  bDone = True
Else
  nLastInputType = wlio.LastInputType
  Select Case nLastInputType
    Case WLCOMMANDTYPE
      Select Case pszVehicleTypeIn
        Case Chr$(24)
          bDone = True
        Case 1      ' F1 Help
          pszVehicleTypeIn = ""
          bStatus = wlerror.ClearError()
          bStatus = wlerror.SetErrorLine("No Help", 0)
          bStatus = wlerror.Display(0)
        Case Else
          bStatus = wlerror.ClearError()
```

```
                bStatus = wlerror.SetErrorLine("Invalid Command", 0)
                bStatus = wlerror.Display(0)
              End Select
            Case WLKEYTYPE

            Case WLSCANTYPE
            Case WLWIDGETTYPE
             Select Case wlio.LastExtendedType
               Case PassCarCheckBox.WidgetID
                  pszVehicleTypeIn = "1"
               Case SUVCheckBox.WidgetID
                 pszVehicleTypeIn = "2"
               Case TruckCheckBox.WidgetID
                  pszVehicleTypeIn = "3"
               Case BoatCheckBox.WidgetID
                  pszVehicleTypeIn = "4"
               Case GVBackButton.WidgetID
                  pszVehicleTypeIn = "5"
               Case GVDoneButton.WidgetID
                  pszVehicleTypeIn = "6"
               Case GVHelpButton.WidgetID
                  pszVehicleTypeIn = ""
                  bStatus = wlerror.ClearError()
                  bStatus = wlerror.SetErrorLine("No Help", 0)
                  bStatus = wlerror.Display(0)
              End Select
              wlcheckboxcoll.ShowWidgets False
        End Select
      End If
    Loop
    Select Case pszVehicleTypeIn
      Case 1      'Passenger Car
        pszModelType = "Passenger Car"
      Case 2      'Sport Utility Vehicle
        pszModelType = "SUV"
      Case 3      'Pickup Truck
        pszModelType = "Pickup Truck"
      Case 4      'Pleasure Boat
        pszModelType = "Pleasure Boat"
      Case 5      'Back to Previous Screen
        bExitGetVehicleType = True
      Case 6      'Done
        bDone = True
        bExitGetVehicleType = True
    End Select
    If bExitGetVehicleType = False Then
```

```
    VerifyData
  End If
End Sub

Public Sub VerifyData()

  Dim VDRow As Integer
  Dim VDColumn As Integer
  Dim nLastInputType As Integer
  Dim pszVerifyInDefault As String
  Dim pszVerifyIn As String
  pszVerifyInDefault = ""
  wlio.RFPrint 0, 0, "WaveLink Corporation", WLCLEAR + WLREVERSE
  wlio.RFPrint 0, 1, "   Auto Detailing   ", WLNORMAL
  wlio.RFPrint 0, 3, " Work Order Detail  ", WLNORMAL
  wlio.RFPrint 0, 5, "Work Type:", WLNORMAL
  wlio.RFPrint 4, 6, pszWorkType, WLNORMAL + WLCLREOLN
  wlio.RFPrint 0, 8, "Vehicle Type:", WLNORMAL
  wlio.RFPrint 4, 9, pszModelType, WLNORMAL + WLCLREOLN
  Select Case nRFTerminalType
    Case WIDGETDEVICE
      VDColumn = 0      'Hide Cursor
      VDRow = 33      'Hide Cursor
      Set VDYesButton = wlfactory.CreateButton(4, _
                        (wlterm.TerminalHeight - 2), 0, 0, _
                        "Accept", wlyesnocoll)
      Set VDNoButton = wlfactory.CreateButton(11, _
                       (wlterm.TerminalHeight - 2), 0, 0, _
                       "Decline", wlyesnocoll)
      wlyesnocoll.StoreWidgets
    Case Else
      VDRow = wlterm.TerminalHeight - 1
      VDColumn = 14
      wlio.RFPrint 0, (wlterm.TerminalHeight - 1), _
           "Correct y/n ? ",  WLREVERSE
  End Select
  pszVerifyIn = wlio.RFInput(pszVerifyInDefault, 1, VDColumn, _
                VDRow, "BarCode", WLCAPSLOCK, WLNO_RETURN_BKSP)
  If nError <> WLNOERROR Then
    bDone = True
  Else
    nLastInputType = wlio.LastInputType
    Select Case nLastInputType
      Case WLCOMMANDTYPE
        wlerror.ClearError
        wlerror.SetErrorLine "Invalid Command", 0
```

```
        wlerror.Display 0
      Case WLWIDGETTYPE
        Select Case wlio.LastExtendedType()
          Case VDYesButton.WidgetID
            pszVerifyIn = "Y"
          Case VDNoButton.WidgetID
            pszVerifyIn = "N"
        End Select
        wlyesnocoll.ShowWidgets False
      Case Else
        If pszVerifyIn = "y" Then
          pszVerifyIn = "Y"
        End If
    End Select
  End If
  If pszVerifyIn = "Y" Then
    GetSignature
  Else
    wlerror.ClearError
    wlerror.SetErrorLine " Work Order Sheet ", 0
    wlerror.SetErrorLine "   has not been   ", 1
    wlerror.SetErrorLine "printed or created", 2
    wlerror.Display 0
  End If

End Sub
Public Sub GetSignature()

  Select Case nRFTerminalType
    Case WIDGETDEVICE
      wlscribble.DisplayDialog "C:\TEMP\x.jpg"
    Case Else
      wlerror.ClearError
      wlerror.SetErrorLine "  Please Have", 0
      wlerror.SetErrorLine " Customer Sign", 1
      wlerror.SetErrorLine "Work Order Sheet", 2
      wlerror.Display 0
  End Select
End Sub

Public Sub Version()

  If bFirstVersion = True Then
  wlio.RFPrint 0, 0, "WaveLink Corporation", WLCLEAR + WLREVERSE
  wlio.RFPrint 0, 1, "   Auto Detailing   ", WLNORMAL
    wlio.RFPrint 0, 3, "Height: " & wlterm.TerminalHeight, WLNORMAL
```

```
    wlio.RFPrint 0, 4, "Width : " & wlterm.TerminalWidth, WLNORMAL
    wlio.RFPrint 0, 5, "Type  : " & nRFTerminalType, WLNORMAL
    wlio.RFPrint 0, 6, "TermID: " & wlterm.TerminalID, WLNORMAL
    wlio.RFPrint 0, 7, "Client: " & wlterm.WaveLinkVersion, _
         WLNORMAL + WLFLUSHOUTPUT
    wlio.PushScreen "Version"
    bFirstVersion = False
  Else
    wlio.RestoreScreen "Version"
  End If
  Select Case nRFTerminalType
    Case WIDGETDEVICE
      Set DoneVerWidget = wlfactory.CreateButton(8, _
                          (wlterm.TerminalHeight - 2), 0, 0, _
                          " Done  ", wlvervidgcoll)
      wlvervidgcoll.StoreWidgets
    Case Else
      wlio.RFPrint 0, (wlterm.TerminalHeight - 1), _
           "    Press any Key    ", WLREVERSE
  End Select
  wlio.GetEvent
  If nRFTerminalType = WIDGETDEVICE Then
    wlvervidgcoll.ShowWidgets False
  End If
End Sub

Public Sub ModNotIn()

  wlerror.ClearError
  wlerror.SetErrorLine "Module  not", 0
  wlerror.SetErrorLine "Implemented", 1
  wlerror.Display 0
End Sub
```

WaveLinkMenubarInfo Object

The WaveLinkMenubarInfo object is supported on: Palm, CE

The WaveLinkMenubarInfo object contains the methods necessary to create and manipulate a list of menus that can be displayed by a menubar widget.

Methods

Clear	Remove
Insert	Replace

Prog IDs

"WaveLink.WaveLinkMenubarInfo"

Remarks

Use the CreateMenubar method of the WaveLinkFactory object to create a menubar widget.

The menu names used by WaveLinkMenubarInfo methods must match the names of menu configurations created using the RFMenu object

Method Summary

Clear Method
- Clears all menu names from the menu list.

Insert Method
- Inserts a menu name at a specific index in the menu list.

Remove Method
- Removes the menu name from a specific index in the menu list.

Replace Method
- Replaces the menu name at a specific index in the menu list.

Clear Method

This member of WaveLinkMenubarInfo is supported on: Palm, CE

The Clear method clears all menu names from the menu list.

VB

```
object.Clear
```

VC++

```
HRESULT hr = object->Clear();
```

Insert Method

This member of WaveLinkMenubarInfo is supported on: Palm, CE

The Insert method inserts a menu name at a specific index position in the menu list.

VB

```
object.Insert Index, Menuname
```

VC++

```
HRESULT hr = object->Insert(short Index, BSTR Menuname);
```

Parameters

Index	The zero-based index for the new menu. Pass an index value of –1 to insert the menu name at the end of the menu list (recommended).
Menuname	The name of the RFMenu configuration

Example

```
' VB Sample Code
Dim myMenInfo As New WaveLinkMenubarInfo
Dim mnuIface As New RFMenu
.
.
.
  'Create the menus that will appear in the main menu bar
  mnuIface.ResetMenu
  mnuIface.AddTitleLine "Widgets Demo's"
  mnuIface.AddOption "Buttons"
  mnuIface.AddOption "Pen Capture"
  mnuIface.StoreMenu "MenOne"
  mnuIface.ResetMenu
  mnuIface.AddTitleLine "Exit"
  mnuIface.AddOption "Quit"
  mnuIface.StoreMenu "MenTwo"
  'Insert the two menus into the WaveLinkMenuBarInfo object,
  'using a -1 as the index automatically
  'tacks the menu to the end of the list
  myMenInfo.Insert -1, "MenOne"
  myMenInfo.Insert -1, "MenTwo"
```

Remove Method

This member of WaveLinkMenubarInfo is supported on: Palm, CE

The Remove method removes the menu name at the specified index.

VB

```
object.Remove Index
```

VC++

```
HRESULT hr = object->Remove(short Index);
```

Parameters

Index	The zero-based index of the menu to be removed.

Replace Method

This member of WaveLinkMenubarInfo is supported on: Palm, CE

The Replace method replaces the menu name at the specified index.

VB

```
object.Replace Index, Menuname
```

VC++

```
HRESULT hr = object->Replace(short Index, BSTR Menuname);
```

Parameters

Index	The zero-based index value of the menu to be replaced.
Menuname	The name of the new RFMenu configuration.

WaveLinkScribblePad Object

The WaveLinkScribblePad object is supported on: Palm, CE

The WaveLinkScribblePad object displays a drawing pad and saves the newly created drawing to a specific (.jpg) file.

Properties

CancelButton
ClearButton
Form
OkButton
Title
ScribblePad

Methods

DisplayDialog

Remarks

Use the ScribblePad object for signature capture in your applications.

Property Summary

CancelButton Property
- Stores and retrieves the pre-defined widget that maintains the cancel button functionality of the scribble pad.

ClearButton Property
- Stores and retrieves the pre-defined widget that maintains the clear button functionality of the scribble pad.

Form Property
- Stores and retrieves the pre-defined widget that maintains the dialog box functionality of the scribble pad.

OkButton Property
- Stores and retrieves the pre-defined widget that maintains the OK button functionality of the scribble pad.

Title Property
- Stores and retrieves the pre-defined widget that maintains the title bar functionality of the scribble pad.

ScribblePad Property
- Stores and retrieves the pre-defined widget that maintains the drawing box functionality of the scribble pad.

Method Summary

DisplayDialog Method

- Displays the scribble pad dialog box and, when the user presses the OK button, saves the picture.

CancelButton Property

This member of WaveLinkScribblePad is supported on: CE

The CancelButton property stores and retrieves a pre-defined WaveLinkWidget object for the WaveLinkScribblePad object. The CancelButton property maintains the cancel functionality of the scribble pad object.

VB

```
WidgName = object.CancelButton

object.CancelButton = WidgName
```

VC++

```
HRESULT hr = object->get_CancelButton(WaveLinkWidget
*WidgName);

HRESULT hr = object->put_CancelButton(WaveLinkWidget
WidgName);
```

Return Values

WidgName	The variable that returns the name of the WaveLinkWidget object

Parameters

WidgName	The variable that stores the name of the WaveLinkWidget object

Remarks

Use the DisplayText property of the WaveLinkWidget object to change the display text that appears on the cancel button. Currently, changing other widget properties for the cancel button will have no effect.

Example

```
' VB Sample Code
Dim myScrib As New WaveLinkScribblePad
Dim myCButton As WaveLinkWidget
.
.
.
myCButton = myScrib.CancelButton
myCButton.DisplayText = "Close"
```

ClearButton Property

This member of WaveLinkScribblePad is supported on: CE

The ClearButton property stores and retrieves a pre-defined WaveLinkWidget object for the WaveLinkScribblePad object. The ClearButton property maintains the "clear drawing" functionality of the scribble pad object.

VB

```
WidgName = object.ClearButton

object.ClearButton = WidgName
```

VC++

```
HRESULT hr = object->get_ClearButton(WaveLinkWidget
*WidgName);

HRESULT hr = object->put_ClearButton(WaveLinkWidget
WidgName);
```

Return Values

WidgName	- The variable that returns the name of the WaveLinkWidget object

Parameters

WidgName	- The name of the WaveLinkWidget object

Remarks

Use the DisplayText property of the WaveLinkWidget object to change the display text that appears on the clear button. Currently, changing other widget properties for the clear button will have no effect.

Form Property

This member of WaveLinkScribblePad is NOT yet implemented on clients.

The Form property stores and retrieves a pre-defined WaveLinkWidget object for the WaveLinkScribblePad object.

VB

```
WidgName = object.Form
object.Form = WidgName
```

VC++

```
HRESULT hr = object->get_Form(WaveLinkWidget *WidgName);
HRESULT hr = object->put_Form(WaveLinkWidget WidgName);
```

Return Values

WidgName	The variable that returns the name of the WaveLinkWidget object

Parameters

WidgName	The name of the WaveLinkWidget object

Remarks

The Form property can be used to change specific characteristics of the dialog box of the scribble pad object.

OkButton Property

This member of WaveLinkScribblePad is supported on: CE

The OkButton property stores and retrieves a pre-defined WaveLinkWidget object for the WaveLinkScribblePad object. The OkButton property maintains the confirmation functionality of the scribble pad object.

VB

```
WidgName = object.OkButton
object.OkButton = WidgName
```

VC++

```
HRESULT hr = object->get_OkButton(WaveLinkWidget *WidgName);
HRESULT hr = object->put_OkButton(WaveLinkWidget WidgName);
```

Return Values

WidgName	- The variable that returns the name of the WaveLinkWidget object

Parameters

WidgName	- The name of the WaveLinkWidget object

Remarks

Use the DisplayText property of the WaveLinkWidget object to change the display text that appears on the Ok button. Currently, changing other widget properties for the Ok button will have no effect.

Example

```
' VB Sample Code
Dim myScrib As New WaveLinkScribblePad
Dim myOkButton As WaveLinkWidget
.
.
.
myOkButton = myScrib.OkButton
myOkButton.DisplayText = "Finish"
```

Title Property

This member of WaveLinkScribblePad is supported on: CE

The Title property stores and retrieves a pre-defined WaveLinkWidget object for the WaveLinkScribblePad object. The Title property maintains the title bar display characteristics of the scribble pad object.

VB

```
WidgName = object.Title

object.Title = WidgName
```

VC++

```
HRESULT hr = object->get_Title(WaveLinkWidget *WidgName);

HRESULT hr = object->put_Title(WaveLinkWidget WidgName);
```

Return Values

WidgName	- The variable that returns the name of the WaveLinkWidget object

Parameters

WidgName	- The name of the WaveLinkWidget object

Remarks

Use the DisplayText property of the WaveLinkWidget object to change the display text that appears on the title bar. Currently, changing other widget properties for the title will have no effect.

Example

```
' VB Sample Code
Dim myScrib As New WaveLinkScribblePad
Dim myTitle As WaveLinkWidget
.
.
.
myTitle = myScrib.Title
myTitle.DisplayText = "Customer Signature"
```

ScribblePad Property

This member of WaveLinkScribblePad is NOT yet implemented on clients.

The ScribblePad property stores and retrieves a pre-defined WaveLinkWidget object for the WaveLinkScribblePad object.

VB

```
WidgName = object.ScribblePad

object.ScribblePad = WidgName
```

VC++

```
HRESULT hr = object->get_ScribblePad(WaveLinkWidget
*WidgName);

HRESULT hr = object->put_ScribblePad(WaveLinkWidget
WidgName);
```

Return Values

WidgName	- The variable that returns the name of the WaveLinkWidget object

Parameters

WidgName	- The name of the WaveLinkWidget object

Remarks

The ScribblePad property can be used to change specific characteristics of the drawing box of the scribble pad object.

DisplayDialog Method

This member of WaveLinkScribblePad is supported on: Palm, CE

The DisplayDialog method displays the scribble pad dialog box. When the user presses the OK button, the picture is saved to a .jpg or .bmp file.

VB

```
object.DisplayDialog FileName
```

VC++

```
HRESULT hr = object->DisplayDialog(BSTR FileName);
```

Parameters

FileName	The file name of the new picture (for example, C:\MyPicture.jpg). If you do not include the full path in the file name, it will be stored in the directory where the application is running.

Example

```
' VB Sample Code
Dim wlscribble As New WaveLinkScribblePad
.
.
.
  wlscribble.DisplayDialog "Sig.jpg"
```

WaveLinkWidget Object

The WaveLinkWidget object is supported on: Palm, CE

The WaveLinkWidget object contains the methods necessary to build and modify all types of widget objects.

Properties

CoordinateType
DisplayBackColor
DisplayFlags
DisplayFontName
DisplayFontSize
DisplayForeColor
DisplayText
Height
InitialFlags
InitialValue
PlatformFlags
SpecialString
WidgetID
WidgetType
Width

Methods

Enable
Focus
SetCoordinates
SetDisplayInfo
SetInitialInfo
SetLabel
SetReturnInfo
SetSpecialInfo
Show

Prog IDs

"WaveLink.WaveLinkWidget"

Remarks

The WaveLinkWidget object is used primarily for manipulating widgets. Although you can also use the WaveLinkWidget object to create widgets, the WaveLinkFactory object effectively automates the widget creation process. For this reason, the WaveLinkFactory object is the recommended method for creating widgets.

Note: Before widgets will display on the RF device, you must use the StoreWidgets method. To return input from a widget, use the RFInput or GetEvent method. These two methods automatically return the widget ID. To process input from the widget based on the widget ID, use the LastExtendedType method. To process input from the widget based on its

general input type (scanned, keyed, widget input, etc.) use the LastInputType method.

Property Summary

CoordinateType Property
- Stores and retrieves the coordinate type of the widget.

DisplayBackColor Property
- Stores and retrieves the background color of the widget.

DisplayFlags Property
- Stores and retrieves the label formatting flags of the widget.

DisplayFontName Property
- Stores and retrieves the font name of the widget.

DisplayFontSize Property
- Stores and retrieves the font size of the widget.

DisplayForeColor Property
- Stores and retrieves the foreground color of the widget.

DisplayText Property
- Stores and retrieves the label display text of the widget.

Height Property
- Stores and retrieves the height of the widget.

InitialFlags Property
- Stores and retrieves the initial state of the widget.

InitialValue Property
- Stores and retrieves the initial value of the widget.

PlatformFlags Property
- Stores and retrieves the platform-specific flags of the widget.

SpecialString Property
- Stores and retrieves information specific to a widget based on its type.

WidgetID Property
- Retrieves the unique identifier of the current widget.

WidgetType Property
- Stores and retrieves the widget type.

Width Property
- Stores and retrieves the width of the widget.

XCoord Property
- Stores and retrieves the starting left coordinate of the widget.

YCoord Property
- Stores and retrieves the starting top coordinate of the widget.

Method Summary

Enable Method
- Enables or disables the widget.

Focus Method
- Sets the input focus to the current widget.

SetCoordinates Method
- Sets the type of positioning, the top left corner, the width, and the height of the widget.

SetDisplayInfo Method
- Sets the formatting and text of the widget.

SetInitialInfo Method
- Sets the initial value and state of the widget.

SetLabel Method
- Changes the display text of the current widget.

SetReturnInfo Method
- Sets the type and value of events returned by the widget.

SetSpecialInfo Method
- Takes in a WaveLinkSpecialInfo object.

Show Method
- Displays or hides the current widget.

CoordinateType Property

This member of WaveLinkWidget is supported on: Palm, CE

The CoordinateType property stores and retrieves the coordinate type for the widget.

VB

```
CoordType = object.CoordinateType

object.CoordinateType = CoordType
```

VC++

```
HRESULT hr = object->get_CoordinateType(CoordinateTypes
*CoordType);

HRESULT hr = object->put_CoordinateType(CoordinateTypes
CoordType);
```

Return Values

CoordType	The variable that retrieves the coordinate type (see Remarks)

Parameters

CoordType	The variable that specifies the coordinate type (see Remarks)

Remarks

The possible values are:

BYCELL	Position the widget using cell coordinates (rows and columns)
BYPIXEL	Position the widget using pixel coordinates
BYPERCENTAGE	Position the widget using percentage coordinates

The CoordinateType property determines how to interpret the values entered in the XCoord, YCoord, Height, and Width properties for the widget. The default value is BYCELL.

DisplayBackColor Property

This member of WaveLinkWidget is supported on: CE

The DisplayBackcolor property stores and retrieves the background color of the widget.

VB

```
DsplBackColor = object.DisplayBackColor

object.DisplayBackColor = DsplBackColor
```

VC++

```
HRESULT hr = object->get_DisplayBackColor(long
*DsplBackColor);

HRESULT hr = object->put_DisplayBackColor(long
DsplBackColor);
```

Return Values

BackColor	The variable that retrieves the background color

Parameters

BackColor	The variable that stores the background color. Pass a value of –1 to use the default system background color

Remarks

The background color must be specified in six-byte RGB format (example, FFFFFF).

DisplayFlags Property

This member of WaveLinkWidget is supported on: Palm, CE

The DisplayFlags property stores and retrieves the label formatting flags of the widget.

VB

```
DsplFlags = object.DisplayFlags
object.DisplayFlags = DsplFlags
```

VC++

```
HRESULT hr = object->get_DisplayFlags(long *DsplFlags);
HRESULT hr = object->put_DisplayFlags(long DsplFlags);
```

Return Values

DsplFlags	The variable that retrieves the label formatting flags (see Remarks)

Parameters

DsplFlags	The variable that specifies the label formatting flags (see Remarks)

Remarks

Valid settings for the label formatting flag include:

CENTERJUST	Center-justified label text
LEFTJUST	Left-justified label text
RIGHTJUST	Right-justified label text

The possible additional formatting flags for a push button widget are:

PBHORIZONTAL	Positions a push button widget horizontally.
PBVERTICAL	Positions a push button widget vertically.
PBPROPORTIONAL	Sizes each box of a push button widget to the size of the text that it contains.
PBFIXED	Sizes each box of a push button widget to the same size.

Example

```
' VB Sample Code
Dim wlwidgcoll As New WaveLinkWidgetCollection
Dim myFactory As New WaveLinkFactory
Dim myWidget As WaveLinkWidget
.
.
.
  ' Create a widget
  Set myWidget = myFactory.CreateButton (1, 2, 0, 0, "Go", _
                 wlwidgcoll)
  ' Set the property to format the widget display
  myWidget.DisplayFlags = CENTERJUST
  wlwidgcoll.StoreWidgets
```

DisplayFontName Property

This member of WaveLinkWidget is supported on: CE

The DisplayFontName property stores and retrieves the font name of the widget.

VB

```
DsplFontName = object.DisplayFontName

object.DisplayFontName = DsplFontName
```

VC++

```
HRESULT hr = object->get_DisplayFontName(BSTR
*DsplFontName);

HRESULT hr = object->put_DisplayFontName(BSTR DsplFontName);
```

Return Values

DsplFontName	The variable that retrieves the font name

Parameters

DsplFontName	The variable that specifies the font name. Pass a value of null to use the default system font.

DisplayFontSize Property

This member of WaveLinkWidget is supported on: CE

The DisplayFontSize property stores and retrieves the font size of the widget.

VB

```
DsplFontSize = object.DisplayFontSize
object.DisplayFontSize = DsplFontSize
```

VC++

```
HRESULT hr = object->get_DisplayFontSize(short
*DsplFontSize);
HRESULT hr = object->put_DisplayFontSize(short
DsplFontSize);
```

Return Values

DsplFontSize	The variable that retrieves the font size

Parameters

DsplFontSize	The variable that specifies the font size. Pass a value of –1 to use the default system font size.

DisplayForeColor Property

This member of WaveLinkWidget is supported on: CE

The DisplayForeColor property stores and retrieves the foreground color of the widget.

VB

```
DsplForeColor = object.DisplayForeColor

object.DisplayForeColor = DsplForeColor
```

VC++

```
HRESULT hr = object->get_DisplayForeColor(long *
DsplForeColor);

HRESULT hr = object->put_DisplayForeColor(long
DsplForeColor);
```

Return Values

DsplForeColor	The variable that retrieves the foreground color

Parameters

DsplForeColor	The variable that specifies the foreground color. Pass a value of –1 to use the default system foreground color.

Remarks

The color must be specified in six-byte RGB format (example, FFFFFF).

DisplayText Property

This member of WaveLinkWidget is supported on: Palm, CE

The DisplayText property stores and retrieves the label display text of the widget.

VB

```
DsplText = object.DisplayText
object.DisplayText = DsplText
```

VC++

```
HRESULT hr = object->get_DisplayText(BSTR *DsplText);
HRESULT hr = object->put_DisplayText(BSTR DsplText);
```

Return Values

DsplText	The variable that retrieves the label display text

Parameters

DsplText	The variable that specifies the label display text

Example

See the OkButton property

Height Property

This member of WaveLinkWidget is supported on: Palm, CE

The Height property stores and retrieves the height of the widget.

VB

```
Height = object.Height
object.Height = Height
```

VC++

```
HRESULT hr = object->get_Height(short *Height);
HRESULT hr = object->put_Height(short Height);
```

Return Values

Height	The variable that retrieves the height

Parameters

Height	The variable that specifies the height (see Remarks)

Remarks

The default setting for the Height and Width property is AUTOSIZE (specified by a value of 0). This setting sizes a button or selector trigger widget according to the size of the text that the it contains. To give a hotspot widget functionality, however, you must set its width and height to something other than than AUTOSIZE. It is recommended that you use the AUTOSIZE setting for the height and width of all other widget types.

The CoordinateType property determines how to interpret numerical values entered for the height of the widget.

InitialFlags Property

This member of WaveLinkWidget is supported on: Palm, CE

The InitialFlags property stores and retrieves the initial state of the widget.

VB

```
InitialFlags = object.InitialFlags
object.InitialFlags = InitialFlags
```

VC++

```
HRESULT hr = object->get_InitialFlags(long *InitialFlags);
HRESULT hr = object->put_InitialFlags(long InitialFlags);
```

Return Values

InitialFlags	The variable that retrieves the initial state (see Remarks)

Parameters

InitialFlags	The variable that specifies the initial state (see Remarks)

Remarks

The possible values are:

INITSTANDARD	Initial state of the widget is standard (shown and enabled)
INITHIDDEN	Initial state of the widget is hidden
INITDISABLED	Initial state of the widget is disabled

A disabled widget will appear on the RF screen, but it will not return when you click on it. The appearance of a disabled widget is platform dependent. Typically, a disabled widget will either be grayed out or a crosshatch pattern will fill its display text.

Note: Unlike other widgets, a disabled hotspot widget will not appear on the RF screen.

Remarks

See the Enable method

InitialValue Property

This member of WaveLinkWidget is supported on: Palm, CE

The InitialValue property stores and retrieves the initial value of the widget.

VB

```
InitialVal = object.InitialValue
object.InitialValue = InitialVal
```

VC++

```
HRESULT hr = object->get_InitialValue(BSTR *InitialVal);
HRESULT hr = object->put_InitialValue(BSTR InitialVal);
```

Return Values

InitialVal	The variable that retrieves the initial value (see Remarks)

Parameters

InitialVal	The variable that specifies the initial value (see Remarks)

Remarks

The possible values for a checkbox widget are:

UNCHECKED
CHECKED
UNDETERMINED

The possible values for a repeater button widget are:

LEFTARROW
RIGHTARROW
UPARROW
DOWNARROW

PlatformFlags Property

This member of WaveLinkWidget is NOT currently implemented on clients.

The PlatformFlags property stores and retrieves the platform-specific flags of the widget object.

VB

```
PfFlags = object.PlatformFlags
object.PlatformFlags = PfFlags
```

VC++

```
HRESULT hr = object->get_PlatformFlags(long *PfFlags);
HRESULT hr = object->put_PlatformFlags(long PfFlags);
```

Return Values

PfFlags	The variable that retrieves the platform-specific flags

Parameters

PfFlags	The variable that specifies the platform-specific flags

SpecialString Property

This member of WaveLinkWidget is supported on: Palm, CE

The SpecialString property specifies information specific to a widget based on its type.

VB

```
SpecialStr = object.SpecialString
object.SpecialString = SpecialStr
```

VC++

```
HRESULT hr = object->get_SpecialString(BSTR *SpecialStr);
HRESULT hr = object->put_SpecialString(BSTR SpecialStr);
```

Return Values

SpecialStr	The variables that retrieves the unique information (see Remarks)

Parameters

SpecialStr	The variables that stores the unique information (see Remarks)

Remarks

For a popup trigger and push button widget, the SpecialString property specifies the name of the RFMenu configuration associated with the widget.

For a bitmap widget, the SpecialString property specifies the bitmap file. The file name must include the full path.

For a menubar widget, the SpecialString property will automatically contain the formatted menu list associated with the widget after you use the SetSpecialInfo method.

WidgetID Property

This member of WaveLinkWidget is supported on: Palm, CE

The WidgetID method retrieves the unique identifer of the current widget

VB

```
WidgID = object.WidgetID
```

VC++

```
HRESULT hr = object->get_WidgetID(long *WidgID);
```

Return Values

WidgID	The variable that retrieves the unique widget identifier

Remarks

The WidgetID property is read-only. The unique widget ID is automatically assigned when you create the widget.

The widget ID is returned through a previous RFInput call or a GetEvent call. To process input from the widget based on the widget ID, use the LastExtendedType method.

To process user input based on the general input type (command key, widget input, scanned input, etc.), use the LastInputType method

Example

```
' VB Sample Code
Dim wlio As New RFIO
Dim DoneWidget As WaveLinkWidget
Dim StartWidget As WaveLinkWidget
Dim nLastInputType As Integer
Dim pszProcessIn As String
pszProcessIn = ""
.
.
. ' In this example, create widgets named StartWidget
  ' and DoneWidget (not shown)
.
.
.
  ' use RFInput or GetEvent to obtain input
  pszProcessIn = wlio.GetEvent

  ' do error checking
.
.
.
  nLastInputType = wlio.LastInputType
  Select Case nLastInputType
    Case WLCOMMANDTYPE
      If pszProcessIn = Chr$(24) Then
        Exit Sub
      End If
    Case WLWIDGETTYPE
      If wlio.LastExtendedType() = StartWidget.WidgetID Then
        GetSignature
      End If
      If wlio.LastExtendedType() = DoneWidget.WidgetID Then
        Exit Sub
      End If
  End Select
```

WidgetType Property

This member of WaveLinkWidget is supported on: Palm, CE

The WidgetType property stores and retrieves the widget type.

VB

```
WidgType = object.WidgetType
object.WidgetType= WidgType
```

VC++

```
HRESULT hr = object->get_WidgetType(short *WidgType);
HRESULT hr = object->put_WidgetType(short WidgType);
```

Return Values

WidgType	The variable that retrieves the widget type (see Remarks).

Parameters

WidgType	The variable that stores the widget type (see Remarks).

Remarks

The possible widget types are:

WAVELINKALLTYPES
WAVELINKBUTTON
WAVELINKHOTSPOT
WAVELINKPOPUP
WAVELINKSELECTOR
WAVELINKPUSHBUTTON
WAVELINKCHECKBOX
WAVELINKREPEATER
WAVELINKLABEL
WAVELINKBITMAP
WAVELINKFIELD
WAVELINKMENUBAR

See the Appendix: Constant Values for the numeric equates returned by the function.

Width Property

This member of WaveLinkWidget is supported on: Palm, CE

The Width property stores and retrieves the width of the widget.

VB

```
Width = object.Width
object.Width = Width
```

VC++

```
HRESULT hr = object->get_Width(short *Width);
HRESULT hr = object->put_Width(short Width);
```

Return Values

Width	The variable that retrieves the width

Parameters

Width	The variable that specifies the width (see Remarks)

Remarks

The default setting for the Width and Height property is AUTOSIZE (specified by a value of 0). This setting sizes a button or selector trigger widget according to the size of the text that it contains. To give a hotspot widget functionality, however, you must set its width and height to something other than than AUTOSIZE. It is recommended that you use the AUTOSIZE setting for the height and width of all other widget types.

The CoordinateType property determines how to interpret numerical values entered for the height of the widget.

XCoord Property

This member of WaveLinkWidget is supported on: Palm, CE

The XCoord property stores and retrieves the starting left coordinate of the widget.

VB

```
LeftCoord = object.XCoord
object.XCoord = LeftCoord
```

VC++

```
HRESULT hr = object->get_XCoord(short *LeftCoord);
HRESULT hr = object->put_XCoord(short LeftCoord);
```

Return Values

LeftCoord	The variable that retrieves the starting left coordinate of the widget

Parameters

LeftCoord	The variable that specifies the starting left coordinate of the widget (see Remarks)

Remarks

The default setting for the XCoord property is 0. The CoordinateType property determines how to interpret values entered for the horizontal position of the widget.

YCoord Property

This member of WaveLinkWidget is supported on: Palm, CE

The YCoord property stores and retrieves the starting upper coordinate of the widget.

VB

```
TopCoord = object.YCoord
object.YCoord = TopCoord
```

VC++

```
HRESULT hr = object->get_YCoord(short *TopCoord);
HRESULT hr = object->put_YCoord(short TopCoord);
```

Return Values

TopCoord	The variable that retrieves the starting top coordinate of the widget

Parameters

TopCoord	The variable that specifies the starting top coordinate of the widget (see Remarks)

Remarks

The default setting for the YCoord property is 0. The CoordinateType property determines how to interpret values entered for the vertical position of the widget.

Enable Method

This member of WaveLinkWidget is supported on: Palm, CE

The Enable method enables or disables the widget.

VB

```
object.Enable EnableWidg
```

VC++

```
HRESULT hr = object->Enable(BOOL EnableWidg);
```

Parameters

EnableWidg	The variable that enables or disables the current widget. A True value enables the widget. A False value disables it.

Remarks

A disabled widget will appear on the RF screen, but it will not return when you click on it. Instead, a crosshatch pattern will fill any display text that the widget contains.

Note: Unlike other widgets, a disabled hotspot widget will not appear on the RF screen.

Example

```
' VB Sample Code
wlwidgcoll As New WaveLinkWidgetCollection
myFactory As New WaveLinkFactory
myWidget As WaveLinkWidget
.
.
.
  ' Create a widget
  Set myWidget = myFactory.CreateButton (1, 2, 0, 0, "Go", _
                 wlwidgcoll)
  ' Set the property to disable the widget
  myWidget.InitialFlags = INITDISABLED
  wlwidgcoll.StoreWidgets
.
.
.
  ' Now enable the disabled widget
  myWidget.Enable True
```

Focus Method

This member of WaveLinkWidget is supported on: Palm

The Focus method sets the input focus to the current widget.

VB

```
object.Focus
```

VC++

```
HRESULT hr = object->Focus();
```

SetCoordinates Method

This member of WaveLinkWidget is supported on: Palm, CE

The SetCoordinates method sets the type of positioning, the top left corner, the width, and the height of the widget.

VB

```
object.SetCoordinates CoordType, LeftCoord, TopCoord,
WidgetWidth, WidgetHeight
```

VC++

```
HRESULT hr = object->SetCoordinates(CoordinateTypes
CoordType, short LeftCoord, short TopCoord, short
WidgetWidth, short WidgetHeight);
```

Parameters

CoordType	The classification of the values placed in LeftCoord, TopCoord, WidgetWidth, and WidgetHeight (BYCELL, BYPIXEL, BYPERCENTAGE)
LeftCoord	The left-most position of the widget
TopCoord	The upper-most position of the widget
WidgetWidth	The horizontal extent of the widget (see Remarks)
WidgetHeight	The vertical extent of the widget (see Remarks)

Remarks

You can assign a special AUTOSIZE setting to the widget by passing a value of 0 for its height and width. This setting sizes a button or selector trigger widget according to the size of the text that it contains. To give a hotspot widget functionality, however, you must set its width and height to something other than than AUTOSIZE. It is recommended that you use the AUTOSIZE setting for the height and width of all other widget types.

The possible coordinate types are:

BYCELL	Position the widget using cell coordinates (rows and columns)
BYPIXEL	Position the widget using pixel coordinates
BYPERCENTAGE	Position the widget using percentage coordinates

SetDisplayInfo Method

This member of WaveLinkWidget is supported on: CE

The SetDisplayInfo method sets the formatting and text of the widget.

VB

```
object.SetDisplayInfo FontName, FontSize, ForeColor,
BackColor, LabelText, DsplFlags
```

VC++

```
HRESULT = object->SetDisplayInfo(BSTR FontName, short
FontSize, long ForeColor, long BackColor, BSTR LabelText,
long DsplFlags);
```

Parameters

FontName	The typeface for the label display
FontSize	The typeface size for the label display
ForeColor	Specifies the color, in six-byte RGB format, of the label text (example, FFFFFF)
BackColor	Specifies the color, in six-byte RGB format, of the widget background
LabelText	The display text of the widget
DsplFlags	The modifiers for the display text (see Remarks)

Remarks

The possible values for the DsplFlags parameter include:

CENTERJUST

LEFTJUST

RIGHTJUST

The possible additional values for the display flags of a push button widget are:

PBHORIZONTAL	Positions a push button widget horizontally.
PBVERTICAL	Positions a push button widget vertically.
PBPROPORTIONAL	Sizes each box of a push button widget to the size of the text that it contains.

PBFIXED	Sizes each box of a push button widget to the same size.

SetInitialInfo Method

This member of WaveLinkWidget is supported on: Palm, CE

The SetInitialInfo method sets the initial value and state of the widget.

VB

```
object.SetInitialInfo InitialVal, InitialState
```

VC++

```
HRESULT = object->SetInitialInfo(BSTR InitialVal, long
InitialState);
```

Parameters

InitialVal	The variable that specifies the initial value (see Remarks)
InitialState	The variable that specifies the initial state (see Remarks)

Remarks

A disabled widget will appear on the RF screen, but it will not return when you click on it. Instead, a crosshatch pattern will fill any display text that the widget contains.

Note: Unlike other widgets, a disabled hotspot widget will not appear on the RF screen.

The possible values for the initial state (InitialState) of the widget are:

INITSTANDARD	Initial state of the widget is standard (shown and enabled)
INITHIDDEN	Initial state of the widget is hidden
INITDISABLED	Initial state of the widget is disabled

The possible values for the initial value (InitialVal) of a checkbox widget are:

CHECKED

UNCHECKED

UNDETERMINED

The possible values for the initial value (InitialVal) of a repeater button widget are:

LEFTARROW

RIGHTARROW

UPARROW

DOWNARROW

SetLabel Method

This member of WaveLinkWidget is supported on: Palm, CE

The SetLabel method changes the display text of the current widget.

VB

```
object.SetLabel NewLab
```

VC++

```
HRESULT hr = object->SetLabel(BSTR NewLab);
```

Parameters

NewLab	The new display text for the current widget

Remarks

This method applies to button, checkbox, and selector trigger widgets.

SetReturnInfo Method

This member of WaveLinkWidget is supported on: Palm, CE

The SetReturnInfo method sets the type and value of events returned by the widget.

VB

```
object.SetReturnInfo Index, EventType, EventValue, Symbology
```

VC++

```
HRESULT hr = object->SetReturnInfo(short Index, short
EventType, BSTR EventValue, short Symbology);
```

Parameters

Index	The variable that specifies the return type for widgets with multiple return types Pass a value of 0 unless the widget has multiple return types (see Remarks).
EventType	The variable that specifies the event type returned by the widget (by default, WLWIDGETTYPE)
EventValue	The variable that specifies the value returned by the widget (by default, the default value is different for different widget types).
Symbology	The variable that specifies the symbology type (see Remarks). Use a value of WLNOSYMBOLOGY if you do not want a barcode symbology returned by the widget.

Remarks

The default return type is WLWIDGETTYPE. To process user input based on the return type (command key, widget input, scanned input, etc.), use the LastInputType method.

The default value returned by a widget is different for different widget types. Use the RFInput or GetEvent method to return the value of the widget.

Use a value of zero (0) for the Index parameter for all widgets unless the widget has multiple return types. Currently, only the checkbox widget has multiple return types. The possible values for the checkbox widget are:

0 - CHECKED

1 - UNCHECKED

2 – UNDETERMINED

The possible symbology types are:

B_UPC
CODABAR
CODE_11
CODE_128
CODE_39
CODE_93
CODE_D25
CODE_I25
D25_IATA
EAN_13
EAN_8
MSI
PDF_417
TO_39
UCC_128
UPC_A
UPC_E0
UPC_E1
WLNOSYMBOLOGY

Example

```
' VB Sample Code
wlwidgcoll As New WaveLinkWidgetCollection
myFactory As New WaveLinkFactory
myWidget As WaveLinkWidget
.
.
.
  ' create one or more widgets
  Set myWidget = myFactory.CreateRepeaterButton (1, 2, 0, 0, _
                 RIGHTARROW, wlwidgcoll)
  'set the return info
  myWidget.SetReturnInfo 0, WLKEYTYPE, "RIGHT", WLNOSYMBOLOGY
  wlwidgcoll.StoreWidgets
```

SetSpecialInfo Method

This member of WaveLinkWidget is supported on: Palm, CE

The SetSpecialInfo method takes in a WaveLinkSpecialInfo object.

VB

```
object.SetSpecialInfo Specialinfo
```

VC++

```
HRESULT hr = object->SetSpecialInfo(WaveLinkSpecialInfo
Specialinfo);
```

Parameters

Specialinfo	The special information associated with the WaveLinkSpecialInfo object.

Remarks

Currently, the only valid WaveLinkSpecialInfo object is the WaveLinkMenubarInfo object.

Although you can use the SetSpecialInfo method to associate a WaveLinkMenubarInfo object with a menubar widget, it is recommended that you use the CreateMenubar method of the WaveLinkFactory object to automate this process.

Show Method

This member of WaveLinkWidget is supported on: Palm, CE

The Show method displays or hides the current widget.

VB

```
object.Show ShowWidg
```

VC++

```
HRESULT hr = object->Show(BOOL ShowWidg);
```

Parameters

ShowWidg	The variable that shows or hides the current widget. A True value shows the widget. A False value hides it.

Remarks

When the ShowWidg parameter is set to True, the Show method will display a widget that has been previously hidden. Before using the Show method, you must use the StoreWidgets method of the WaveLinkWidgetCollection object to store widgets on the RF device while simultaneously causing them to display on the RF screen. The StoreWidgets method will not display widgets whose initial state is set to hidden.

Example

```
' VB Sample Code
wlwidgcoll As New WaveLinkWidgetCollection
myFactory As New WaveLinkFactory
myWidget As WaveLinkWidget
.
.
.
  ' Create a widget
  Set myWidget = myFactory.CreateRepeaterButton (1, 2, 0, 0, _
                 RIGHTARROW, wlwidgcoll)
  ' Set the property to hide the widget
  myWidget.InitialFlags = INITHIDDEN
  wlwidgcoll.StoreWidgets
.
.
.
  ' Now show the hidden widget
  myWidget.Show True
```

WaveLinkWidgetCollection Object

The WaveLinkWidgetCollection object is supported on: Palm, CE

The WaveLinkWidgetCollection object logically groups widget objects together.

Properties

Count

Methods

Add
Clear
DeleteAllWidgets
DeleteWidgets
EnableAllWidgets
EnableWidgets
Item
Remove
ShowAllWidgets
ShowWidgets
StoreWidgets

Prog IDs

"WaveLink.WaveLinkWidgetCollection"

Remarks

Use the WaveLinkWidgetCollection object to manipulate a group of widgets, such as a set of widgets that appear on the same screen.

Property Summary

Count Property
- Retrieves the number of widget objects in the collection.

Method Summary

Add Method
- Adds a widget to the current collection.

Clear Method
- Clears all widgets from the current collection.

DeleteAllWidgets Method
- Deletes all widgets present on the RF device.

DeleteWidgets Method
- Deletes all widgets in the collection from the RF device.

EnableAllWidgets Method
- Enables or disables all widgets present on the RF device.

EnableWidgets Method
- Enables or disables all widgets in the current collection.

Item Method
- Returns the widget specified by the ID or index value from the current collection.

Remove Method
- Removes a specific widget based on its index value.

ShowAllWidgets Method
- Shows or hides all widgets present on the RF device.

ShowWidgets Method
- Shows or hides all widgets in the current collection.

StoreWidgets Method
- Stores all widgets in the collection and displays them on the RF device.

Count Property

This member of WaveLinkWidgetCollection is supported on: Palm, CE

The Count property retrieves the number of widget objects contained in the WaveLinkWidget collection.

VB

```
Items = object.Count
```

VC++

```
HRESULT hr = object->getCount(long *Items);
```

Return Values

Items	The number of widgets in the collection.

Remarks

The Count property is a read-only property.

Members of the collection loop starting with the member number 1 and ending with the value of the Count property. If you want to loop through the members of the collection without checking the Count property, you can use a "For Each…Next" command. If it returns a value of 0, the collection contains no objects.

Add Method

This member of WaveLinkWidgetCollection is supported on: Palm, CE

The Add method adds a widget to the current collection.

VB

```
object.Add NewWidget
```

VC++

```
HRESULT hr = object->Add(IWaveLinkWidget NewWidget);
```

Parameters

NewWidget	The widget object to be added to the collection

Clear Method

This member of WaveLinkWidgetCollection is supported on: Palm, CE

The ClearWidgets method clears all widgets within the current object.

VB

```
object.ClearWidgets
```

VC++

```
HRESULT hr = object->ClearWidgets();
```

DeleteAllWidgets Method

This member of WaveLinkWidgetCollection is supported on: Palm, CE

The DeleteAllWidgets method deletes all widgets present on the RF device.

VB

```
object.DeleteAllWidgets
```

VC++

```
HRESULT hr = object->DeleteAllWidgets();
```

DeleteWidgets Method

This member of WaveLinkWidgetCollection is supported on: Palm, CE

The DeleteWidgets method deletes all widgets within the current collection from the RF device.

VB

```
object.DeleteWidgets
```

VC++

```
HRESULT hr = object->DeleteWidgets();
```

EnableAllWidgets Method

This member of WaveLinkWidgetCollection is supported on: Palm, CE

The EnableAllWidgets method causes all widgets present on RF device to be enabled or disabled.

VB

```
object.EnableAllWidgets EnableWidgs
```

VC++

```
HRESULT hr = object->EnableAllWidgets(BOOL EnableWidgs);
```

Parameters

EnableWidgs	The variable that enables or disables the widgets. A True value enables the widgets. A False value disables them.

EnableWidgets Method

This member of WaveLinkWidgetCollection is supported on: Palm, CE

The EnableWidgets method causes all widgets within the current collection to be enabled or disabled.

VB

```
object.EnableWidgets EnableWidgs
```

VC++

```
HRESULT hr = object->EnableWidgets(BOOL EnableWidgs);
```

Parameters

EnableWidgs	The variable that enables or disables the widgets. A True value enables the widgets; a False value disables them.

Example

```
' VB Sample Code
wlwidgcoll As New WaveLinkWidgetCollection
myFactory As New WaveLinkFactory
myWidget As WaveLinkWidget
.
.
.
  ' Create one or more widgets
  Set myWidget = myFactory.CreateButton (1, 2, 0, 0, "Go", _
                 wlwidgcoll)
  ' Set the property to disable the widget(s)
  myWidget.InitialFlags = INITDISABLED
  wlwidgcoll.StoreWidgets
.
.
.
  ' Now enable the disabled widget(s)
  wlwidgcoll.EnableWidgets True
```

Item Method

This member of WaveLinkWidgetCollection is supported on: Palm, CE

The Item method returns the widget specified by the ID or index value from the current collection object.

VB

```
Widget = object.Item(ID)
```

VC++

```
HRESULT hr = object->Item(VARIANT ID, LPVARIANT Index);
```

Return Values

Widget	The variable that receives a reference to a widget object

Parameters

ID	The widget ID or the index value of the widget. To pass the ID, you must enclose it in quotation marks (for example, "4"). To pass the index value, you must not enclose the value in quotation marks (for example, 4).

Remarks

Note to VB users: The WaveLinkWidget collection executes the Item method by default if another property or method is not specified in place of it.

Remove Method

This member of WaveLinkWidgetCollection is supported on: Palm, CE

The RemoveWidget method removes the specified widget from the collection based on the index value of the widget.

VB

```
object.RemoveWidget Index
```

VC++

```
HRESULT hr = object->RemoveWidget(long Index);
```

Parameters

Index	The zero-based index value of the widget to be removed.

ShowAllWidgets Method

This member of WaveLinkWidgetCollection is supported on: Palm, CE

The ShowAllWidgets method causes all widgets present on the RF device to be shown or hidden.

VB

```
object.ShowAllWidgets ShowWidgs
```

VC++

```
HRESULT hr = object->ShowAllWidgets(BOOL ShowWidgs);
```

Parameters

ShowWidgs	The variable that shows or hides the widgets. A True value shows the widgets. A False value hides them.

Remarks

When the ShowWidg parameter is set to True, the ShowAllWidgets method will display a widget that has been previously hidden. Before using the ShowAllWidgets method, you must use the StoreWidgets method to store widgets on the RF device while simultaneously causing them to display on the RF screen. The StoreWidgets method will not display widgets whose initial state is set to hidden.

ShowWidgets Method

This member of WaveLinkWidgetCollection is supported on: Palm, CE

The ShowWidgets method causes all widgets within the current collection to be shown or hidden.

VB

```
object.ShowWidgets ShowWidgs
```

VC++

```
HRESULT hr = object->ShowWidgets(BOOL ShowWidgs);
```

Parameters

ShowWidgs	The variable that shows or hides the widgets. A True value shows the widgets. A False value hides them.

Remarks

When the ShowWidg parameter is set to True, the ShowWidgets method will display a widget that has been previously hidden. Before using the ShowWidgets method, you must use the StoreWidgets method to store widgets on the RF device while simultaneously causing them to display on the RF screen. The StoreWidgets method will not display widgets whose initial state is set to hidden.

Example

```
' VB Sample Code
Dim wlwidgcoll As New WaveLinkWidgetCollection
.
.
.
  ' Create one or more widgets, add them to the
  ' wlwidgcoll collection, store them (not shown)
.
.
.
  ' Hide the widgets
  wlwidgcoll.ShowWidgets False
```

StoreWidgets Method

This member of WaveLinkWidgetCollection is supported on: Palm, CE

The StoreWidgets method stores all widget objects in the collection. Widgets whose initial state is set to standard will display immediately when you use the StoreWidgets method.

VB

```
object.StoreWidgets
```

VC++

```
HRESULT hr = object->StoreWidgets();
```

Remarks

The initial state of a widget is standard by default. Use the InitialFlags property of the WaveLinkWidget object to change the initial state.

Example

```
' VB Sample Code
Dim wlio As New RFIO
Dim wlfactory As New WaveLinkFactory
Dim wlwidgcoll As New WaveLinkWidgetCollection
Dim OKWidget As WaveLinkWidget
.
.
.
  ' In this example, give the widget some context
  wlio.RFPrint 0, 0, "WaveLink Corporation", WLCLEAR + WLREVERSE
  wlio.RFPrint 0, 1, "   Auto Detailing   ", WLNORMAL
  ' Create the widget
  Set OKWidget = wlfactory.CreateButton(4, _
                 (wlterm.TerminalHeight - 2), _
                 0, 0, "  OK  ", wlwidgcoll)
.
.
.
  wlwidgcoll.StoreWidgets
```

Appendix A: Constant Values

This appendix contains the numeric equates for the WaveLink constants.

The constants associated with the RFGetLastError methods are classified as error constants. The remaining WaveLink constants have been classified according to the specific object with which they are associated:

- Error Constants
- RFAuxPort Constants
- RFBarcode Constants
- RFIO Constants
- RFTerminal Constants
- WaveLinkWidget Constants

Error Constants

Table 1: *Error Constants*

Constant	Value	Description
WLNOERROR	0	No error
WLCOMMERROR	1	Communication error
WLMEMORYERROR	2	Memory allocation error
WLNOTINITIALIZED	3	RF connection not initialized
WLREQUESTFAIL	4	Input request error
WLINVALIDPARAMS	5	Invalid function parameters
WLINNVALIDRETURN	6	Invalid RF function return packet
WLFUNCTIONFAILED	7	RF function call failed at RF device
WLBCPARSEFAILURE	8	Failure parsing barcode file
WLBCADDERROR	9	Failure adding barcode configuration
WLBCCLEARERROR	10	Failure clearing barcode config list
WLBARCODELIMITEXCEEDED	11	Exceeded number of allowable barcode configs

Table 1: *Error Constants*

Constant	Value	Description
WLTONEADDERROR	12	Failure adding tone configuration
WLTONECLEARERROR	13	Failure clearing tone config list
WLIOQUEUEERROR	14	IO queue buffer error
WLNOIPUTTYPE	15	Unable to determine input type
WLPORTTIMEOUT	16	Aux port timeout
WLDTOUTOFSYNC	17	Terminal Date/Time is out of sync by more that 5 seconds

RFAuxPort Constants

Table 2: *RFAuxPort Constants*

Constant	Value	Description
WLCOM1	0	Com port 1
WLCOM2	1	Com port 2
WLNOCHAR'	'\0'	No start/end character
WLNOMAXLENGTH	-1	No query max length
WLWAITFOREVER	-1	No query timeout
BAUD150	0	150 baud connection
BAUD300	1	300 baud connection
BAUD600	2	600 baud connection
BAUD1200	3	1200 baud connection
BAUD1350	4	1350 baud connection
BAUD2400	5	2400 baud connection
BAUD4800	6	4800 baud connection
BAUD9600	7	9600 baud connection
BAUD19200	8	19200 baud connection
BAUD38400	9	38400 baud connection
BAUD110	10	110000 baud connection

Table 2: *RFAuxPort Constants*

Constant	Value	Description
DATABITS5	0	5 data bits
DATABITS6	1	6 data bits
DATABITS7	2	7 data bits
DATABITS8	3	8 data bits
DATABITS4	4	4 data bits
PARITYEVEN	0	Even parity
PARITYMARK	2	Mark parity
PARITYNONE	4	None parity
PARITYODD	1	Odd parity
PARITYSPACE	3	Space parity
STOPBITS1	0	One stop bit
STOPBITS2	1	Two stop bits
HARDWAREFLOWCTL	2	Hardware flow control
NOFLOWCTL	0	No flow control
SOFTWAREFLOWCTL	1	Software flow control

RFBarcode Constants

Table 3: *RFBarcode Constants*

Constant	Value	Description
WLNOSYMBOLOGY	99	No barcodes acceptable for input

RFIO Constants

Table 4: *RFIO Constants*

Constant	Value	Description
WLCLEAR	4	Clear screen before output
WLCLREOLN	8	Clear to end of line
WLCLREOS	32	Clear to end of screen
WLFLUSHOUTPUT	16	Flush the output buffer immediately
WLNORMAL	1	Normal video mode
WLREVERSE	2	Reverse video mode
WLCOMMANDTYPE	2	Command key entered
WLINPUTERROR	0	Error from input call
WLKEYTYPE	4	Normal key entered
WLMENUBARTYPE	6	Input from a menubar widget
WLPOPUPTRIGGER	7	Input from a popup trigger widget
WLSCANTYPE	5	Scanned data entered
WLWIDGETTYPE	8	Input from a widget
WLTIMEDOUT	9	RFInput input prompt expired
WLCAPSLOCK	64	Set caps lock for input
WLNORMALKEYS	0	Normal keyboard mode
WLALPHA_ONLY	0x0080	Accept alpha characters only
WLBACKLIGHT	0x0200	Enable display backlight
WLCLR_INPUT_BUFFER	0x2000	Clear the input buffer of extra data
WLDISABLE_FKEYS	0x0020	Disable function key input
WLDISABLE_KEY	0x0002	Disable keyboard input
WLDISABLE_SCAN	0x0001	Disable scanner on input
WLECHO_ASTERISK	0x8000	Echo asterisk for each character
WLFORCE_ENTRY	0x0040	Do not return with empty input field

Table 4: *RFIO Constants*

Constant	Value	Description
WLINCLUDE_DATA	0x0800	Include data with function key entry
WLINGORE_CRLF	0x4000	Do not break packages on <CR> or <LF>
WLMAXLENGTH	0x0400	Do not accept more than nLength characters
WLNO_NONPRINTABLE	0x01000 0	Supress non-printable characters
WLNO_RETURN_BKSP	0x0010	Do not return on backspace
WLNO_RETURN_FILL	0x0008	Do not return when input prompt length is reached
WLNUMERIC_ONLY	0x0100	Accept numeric keys only
WLSOFT_TRIGGER	0x1000	Soft RF device trigger
WLSUPRESS_ECHO	0x0004	Supress input echo

RFTerminal Constants

Table 5: *RFTerminal Constants*

Constant	Value	Description
CAPSLOCK	64	Keyboard mode is capslock
NORMALKEYS	0	Keyboard mode is normal
HARDWARECURSOR	0	Use hardware input cursor
SOFTWARECURSOR	1	Use software input cursor
LITHIUMBATDEAD	0	RF device's lithium battery is dead
LITHIUMBATGOOD	1	RF device's lithium battery is good
LITHIUMBATNONE	2	RF device's lithium battery is not present

Table 5: *RFTerminal Constants*

Constant	Value	Description
MAINBATGOOD	0	RF device's main battery is good
MAINBATLOW	1	RF device's main battery is low

WaveLinkWidget Constants

Table 6: *WaveLink Widget Constants*

Constant	Value	Description
AUTOSIZE	0	Determine the widget extent automatically
BYCELL	2	Position using cell coordinates
BYPERCENTAGE	3	Position using percentage coordinates
BYPIXEL	1	Position using pixel coordinates
CENTERJUST	0x00000 4	Center justify the label
LEFTJUST	0x00000 1	Left justify the label
RIGHTJUST	0x00000 2	Right justify the label
CHECKED	1	Checked checkbox state
UNCHECKED	0	Unchecked checkbox state
UNDETERMINED	2	Undetermined checkbox state
INITDISABLED	2	Initial state is Disabled
INITHIDDEN	1	Initial state is Hidden
INITSTANDARD	0	Initial state of the widget is standard
PBFIXED	0x00004 0	Size each box of push button equally

Table 6: *WaveLink Widget Constants*

Constant	Value	Description
PBHORIZONTAL	0x00000 8	Position push button horizontally
PBPROPORTIONAL	0x00002 0	Size each box of push button to size of its text
PBVERTICAL	0x00001 0	Position a push button vertically
DOWNREPEATER	4	Down arrow repeater button
LEFTREPEATER	1	Left arrow repeater button
RIGHTREPEATER	2	Right arrow repeater button
UPREPEATER	3	Up arrow repeater button
WAVELINKBITMAP	9	Bitmap image widget
WAVELINKBUTTON	1	Button widget
WAVELINKCHECKBOX	6	Checkbox widget
WAVELINKFIELD	10	Field widget. This widget is an input box.
WAVELINKHOTSPOT	2	Hotspot widget
WAVELINKLABEL	8	Label widget
WAVELINKMENUBAR	12	Menubar widget it
WAVELINKPOPUP	3	Popup trigger widget
WAVELINKPUSHBUTTON	5	Push button widget
WAVELINKREPEATER	7	Repeater button widget
WAVELINKSELECTOR	4	Selector trigger widget

Index

S

T

V

W

X

Y