



**BancTec Limited**  
**RFeasyServ Programmers Manual**

**Copyright © Jan 2006**

# OVERVIEW

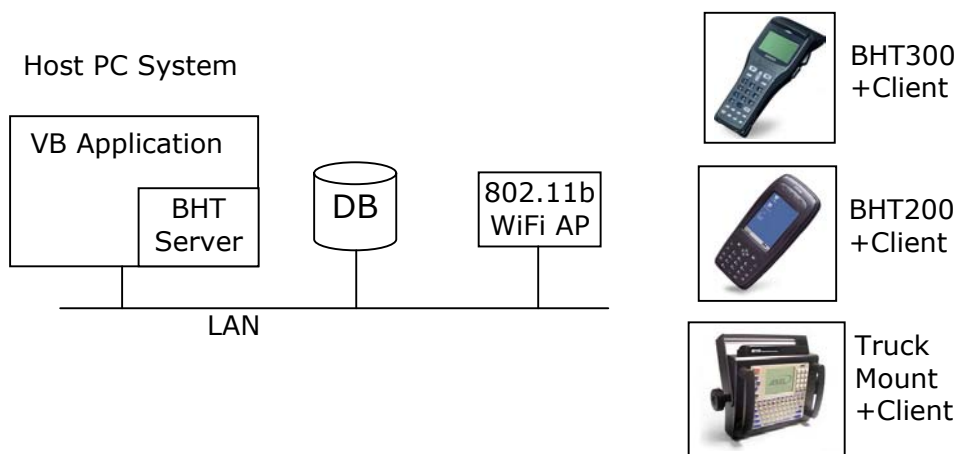
The RFeasyServ control (**RFeasyServ.ocx**) from BancTec is a 32-bit Windows ActiveX component that manages the user screens and data flow between a mobile terminal and a host application via a standard 802.11b WiFi network.

The control has been designed to assist developers in creating powerful RF client-server applications quickly and easily without having to get bogged down in the RF communications. The control provides a simple set of methods and events for the developer to control everything on the HHT and get notification of an input. These include screen outputs, barcode & keyed input fields, RS232 I/O to peripherals, LED and buzzer control, form/frame management and flow control etc.

The control on the Server coupled with the Client on the mobile terminal takes care of all the low-level communications, packet compilation, flow control etc allowing the developer to concentrate on the application functionality.

The Client will be available on a range of products from different manufacturers all connecting to the same RFeasyServ control and thus same Host Application. These will include the **DENSO** BHT300, BHT7500, BHT200 and BHT400 units plus a number of other CE.Net & XP devices including truck mount units, PDAs etc.

As all Client to Server links are the same it is possible to mix the mobile clients to suit your needs. e.g. handheld terminals for picking and goods in, Truck Mount units for bulk transfers and despatch etc..



The RFeasyServ control can be used in any application that supports ActiveX controls, Visual Studio, Visual Basic, Visual C++, Excel, Access, FoxPro, Delphi.

This manual describes the methods and events of the RFeasyServ control. For other references please refer to the RFeasyServ programmers manual and the sample source code.

# METHODS

---

**Ack** ( *UnitID* As Long ) As Boolean

The Ack command is added to the transmit buffer for this UnitID. This is used with the **NewField()** command and the FLD\_CLR style OR to Acknowledge a transmission.

**Parameters:**

<i>UnitID</i>	HHT serial number
---------------	-------------------

**Return Value:**

**TRUE** if the command has been added to the transmit buffer for this UnitID  
**FALSE** if transmit buffer for this UnitID is full

**See Also:**

**NewField()**

---

**Beep** ( *UnitID* As Long, *OnTime* As Integer, *Delay* As Integer ) As Boolean

The method adds a Beep command to the transmit buffer for this UnitID, this generates a beep of specific length (OnTime) and delay (Delay) on the HHT

**Parameters:**

<i>UnitID</i>	HHT serial number
<i>OnTime</i>	Beeper is ON 10 x <i>length</i> milliseconds (eg. value 100 = 1 seconds)
<i>Delay</i>	Beeper is OFF 10 x <i>delay</i> milliseconds

**Example:**

SOS tone

```
Beep(unitid, 5, 10) 'Beep 50ms ON and 100 ms OFF
Beep(unitid, 5, 10)
Beep(unitid, 5, 30)
Beep(unitid, 15, 10) 'Beep 150ms ON and 100 ms OFF
Beep(unitid, 15, 10)
Beep(unitid, 15, 30)
Beep(unitid, 5, 10) 'Beep 50ms ON and 100 ms OFF
Beep(unitid, 5, 10)
Beep(unitid, 5, 30)
```

**Return Value:**

**TRUE** if the command has been added to the transmit buffer for this UnitID  
**FALSE** if transmit buffer for this UnitID is full

**See Also:**

**Bell()**

---

**Bell** ( *UnitID* As Long ) As Boolean

This method adds a Bell command to the transmit buffer for this UnitID, which generates a simple beep on the HHT

**Parameters:**

<i>UnitID</i>	HHT serial number
---------------	-------------------

**Return Value:**

**TRUE** if the command has been added to the transmit buffer for this UnitID  
**FALSE** if transmit buffer for this UnitID is full

**See Also:**

**Beep()**

---

**Button** ( *UnitID* As Long, *StartPos* As Integer, *ButtonData* As String [, *LargeText* As Boolean = False] ) As Boolean

This method creates a "button" style field for this UnitID, which is a locked field with locked text. When the field is highlighted and the ENT key pressed then the text on the button will be sent to the host.

Using several button fields on a screen can produce an easy menu.

If you need other input fields to return data as well as a button style field then use the **NewField( )** and **FldTxt( )** methods with SND\_ALL and NO\_SEND options.

**Parameters:**

<i>UnitID</i>	HHT serial number
<i>StartPos</i>	Position of the input field
<i>ButtonData</i>	Text for the button
<i>LargeText</i> ( <i>Optional</i> )	<b>TRUE</b> Uses double height font <b>FALSE</b> (default) Uses normal height font

**Return Value:**

**TRUE** if the command has been added to the transmit buffer for this UnitID  
**FALSE** if transmit buffer for this UnitID is full

**See Also:**

**NewField( )**  
**FldTxt( )**

---

**ClearCmd** ( *UnitID* As Long, *Command* As Integer, *StartPos* As Integer, *StopPos* As Integer )  
As Boolean

This method will clear the HHT display of any text or field within the specified region.

**Parameters:**

<i>UnitID</i>	HHT serial number		
<i>Command</i>	Value	SYNTAX	DESCRIPTION
	<b>&amp;h01</b>	CLEAR_TXT	Removes text from <i>StartPos</i> to <i>StopPos</i>
	<b>&amp;h02</b>	CLEAR_FLD	Removes fields from <i>StartPos</i> to <i>StopPos</i>
	<b>&amp;h04</b>	CLEAR_FLDDATA	Clears the field data from <i>StartPos</i> to <i>StopPos</i>
<i>StartPos</i>	Start position of the display (0-239)		
<i>StopPos</i>	Stop position of the display (0-239)		

**Return Value:**

**TRUE** if the command has been added to the transmit buffer for this UnitID  
**FALSE** if transmit buffer for this UnitID is full

---

**ClearForm** ( *UnitID* As Long ) As Boolean

This method clears the HHT display of text and fields and sets the cursor to top left position.

**Parameters:**

<i>UnitID</i>	HHT serial number
---------------	-------------------

**Return Value:**

**TRUE** if the command has been added to the transmit buffer for this UnitID  
**FALSE** if transmit buffer for this UnitID is full

---

**ConnectIP** ( *IPAddr* As String, *Port* As Long ) As Boolean

This method opens the connection path between the RFeasyServ control and the NIC of the host computer. The Control will now use this path as the interface between the application and the RF handheld terminal.

**Parameters:**

<i>IPAddr</i>	IP address of the host computer
<i>Port</i>	The local port of the host computer

**Example:**

```
ConnectIP("192.168.1.1", 81)
```

**Return Value:**

**TRUE** if connected successfully  
**FALSE** if the address or port are not available

**See Also:**

**DisconnectIP( )**

---

**DataToSerial** ( *UnitID* As Long, *Data* As String ) As Boolean

This method allows data to be sent to the serial port of the handheld terminal for this UnitID. This can be used for portable printers, card readers or any RS232 input device.

Currently the RS232 serial settings are set: 19200,8,1,None

**Parameters:**

<i>UnitID</i>	HHT serial number
<i>Data</i>	Any ASCII characters (0-255)

**Return Value:**

**TRUE** if the command has been added to the transmit buffer for this UnitID  
**FALSE** if transmit buffer for this UnitID is full

**See Also:**

**WaitSerial( )**

---

**DisconnectIP** ( ) As Boolean

This method closes the connection path between the RFeasyServ control and the NIC of the host computer, thus closing any link to RF handheld terminals.

**Return Value:**

**TRUE** if disconnect was successful

**FALSE** if disconnect failed

**See Also:**

**ConnectIP()**

---

**ErrorBeepLED** ( *UnitID* As Long ) As Boolean

This method adds a warning message indicator for this UnitID HHT, which will light the Red LED and emit a loud beep for 2 seconds

**Parameters:**

<i>UnitID</i>	HHT serial number
---------------	-------------------

**Return Value:**

**TRUE** if the command has been added to the transmit buffer for this UnitID

**FALSE** if transmit buffer for this UnitID is full

**See Also:**

**Beep()**

**BELL()**

**LED()**

---

**FieldCmd** ( *UnitID* As Long, *StartPos* As Integer, *Command* As Integer ) As Boolean

This method is used to add definition to an existing field for this UnitID.

**Parameters:**

<i>UnitID</i>	HHT serial number		
<i>StartPos</i>	Position of the input field		
<i>Command</i>	Value	SYNTAX	DESCRIPTION
	<b>&amp;h01</b>	FLD_REMOVE	Removes the field
	<b>&amp;h02</b>	FLD_CLEAR	Clears the fields (from locked fields also)
	<b>&amp;h04</b>	FLD_LOCK	Locks the field (cannot be written to)
	<b>&amp;h08</b>	FLD_ACTIVE	Field is set to active

**Return Value:**

**TRUE** if the command has been added to the transmit buffer for this UnitID  
**FALSE** if transmit buffer for this UnitID is full

---

**FldTxt** ( *UnitID* As Long, *StartPos* As Integer, *FieldText* As String ) As Boolean

This method populates an input field at position *StartPos* with the *FieldText* for this UnitID. If the *FieldText* is longer than the field any excess text will be ignored and any text already in the field will be replaced. If a field does not exist for this *StartPos* the command will be ignored.

**Parameters:**

<i>UnitID</i>	HHT serial number
<i>StartPos</i>	Position of the input field
<i>FieldText</i>	Text for the field

**Return Value:**

**TRUE** if the command has been added to the transmit buffer for this UnitID  
**FALSE** if transmit buffer for this UnitID is full

---

**GetBatteryLevel** ( *UnitID* As Long ) As Integer

This method will return the latest battery level for this UnitID, which allows the host application to monitor HHTs and perhaps issue a low battery-warning message when the battery goes below a certain level

**Parameters:**

<i>UnitID</i>	HHT serial number
---------------	-------------------

**Return Value:**

Integer value between 0 and 9 (where 0 is 0%-10% and 9 is 90%-100% of battery remaining)

---

**GetData** ( *UnitID* As Long, *Pos* As Integer ) As String

This is the main function to extract the field data returned from the HHT. Once the **DataArrived()** event has been triggered use the **IsData()** and **GetData()** functions to test for and extract data from a field. This function will return the string of data from the field at position Pos. If there is no field at Pos a null string is returned.

**Parameters:**

<i>UnitID</i>	HHT serial number
<i>Pos</i>	Position of the input field (0-239)

**Return Value:**

String data in the field at the specified position.

**See Also:**

**IsData( )**  
**DataArrived( )**  
**GetMark( )**

---

### **GetFormID** ( *UnitID* As Long ) As Integer

This method returns the current form ID that was set by the **SetFormID( )** method. Once the **DataArrived()** event has been triggered use this function to determine which form the data has come from and thus what field data to extract and what to do with this data.

If the form ID has not been set, or the HHT has been reset, then the FormID is returned as 0.

#### **Parameters:**

<i>UnitID</i>	HHT serial number
<i>Pos</i>	Position of the input field (0-239)

#### **Return Value:**

The number of the form defined by **SetFormID()** method.

#### **See Also:**

**SetFormID( )**  
**DataArrived()**

---

### **GetLastFrameID** ( *UnitID* As Long ) As Integer

Each time the **Send()** method is used to transmit the buffer to the HHT it is assigned a frameid. Use this method to check what the last frame to be sent was.

#### **Parameters:**

<i>UnitID</i>	HHT serial number
---------------	-------------------

#### **Return Value:**

Last sent frameid (**Send()** method parameter 2).

#### **See Also:**

**Send()**

---

**GetMark** ( *UnitID* As Long, *Pos* As Integer ) As String

This function is used to return barcode symbology and length of a data field. Once the **DataArrived()** event and **IsData()** and **GetData()** have been used to determine and retrieve data then this function can be used to return details on the barcode type & length. If data field was not scanned then this function will return a blank string.

**Parameters:**

<i>UnitID</i>	HHT serial number
<i>Pos</i>	Position of the input field (0-239)

**Return Value:**

X99 Where X = Barcode symbology (see below) and 99 = barcode length

e.g. A13 = 13 digit EAN 13 code.

Symbology marks are

A = EAN-13, UPC-A

B = EAN-8

C = UPC-E

I = ITF (Interleaved 2of5)

H = STF (Standard 2of5)

N = Codabar (NW-7)

M = Code 39

L = Code 93

K = Code 128

W = EAN-128

**See Also:**

**IsData( )**

**DataArrived( )**

---

**GetMessageNumber** ( *UnitID* As Long ) As Integer

Every time a packet is sent to the HHT a new message number is assigned. Use this function to return the message number of the HHT, the message number is between 0 and 15.

**Parameters:**

<i>UnitID</i>	HHT serial number
---------------	-------------------

**Return Value:**

Message number (0 - 15).

---

**GetSerialData** ( *UnitID* As Long ) As String

Retrieves data from the serial port buffer after the port has been opened using the **WaitSerial()** method, with the duration of 255. The host system can then process the data without displaying it.

**Parameters:**

<i>UnitID</i>	HHT serial number
---------------	-------------------

**Example:****Using the GetSerialData() method:**

```
Sub CreateForm() 'This routine defines the forms objects
...
RFeasyServ.WaitSerial(UnitID, 255) 'Opens the serial port using the
duration value 255
RFeasyServ.Send(UnitID, FrameID)
...
End Sub

Sub ProcessForm() 'This routine processes data from the form
...
mydata = RFeasyServ.GetSerialData(UnitID) 'Gets the serial data
...
End Sub
```

Data from the serial port goes directly back to the host application where the data is processed

**Return Value:**

Data from the handheld terminal serial port.

**See Also:**

**WaitSerial()**

---

**IsData** ( *UnitID* As Long, *Pos* As Integer ) As Boolean

Checks position *Pos* on the current form to see if there is data at that position, if there is the method returns **TRUE**, if there is no data it will return **FALSE**.

**Parameters:**

<i>UnitID</i>	HHT serial number
<i>Pos</i>	Position of the input field (0 - 239)

**Example:**

```
Sub ProcessForm() 'This routine defines the forms objects
...
If RFeasyServ.IsData(UnitID, Pos) = True Then 'Check if there is data
on the form
RFeasyServ.GetData(UnitID, Pos) 'Retrieve data from position
End If
...
End Sub
```

**Return Value:**

**TRUE** If there is data sent from the input field at position *Pos*  
**FALSE** If there is no data at position *Pos*

**See Also:**

**GetData()**

---

**LED** ( *UnitID* As Long, *Colour* As Integer, *Duration* As Integer ) As Boolean

Turns on the LED a colour for a specific duration.

**Parameters:**

<i>UnitID</i>	HHT serial number
<i>Colour</i>	RED = 1 GREEN = 2
<i>Duration</i>	Light is ON 100 x <i>duration</i> milliseconds (eg. value 10 = 1 seconds)

**Example:**

```
LED(unitid, 1, 10) 'Lights LED red for 1 second

LED(unitid, 2, 30) 'Lights LED green for 3 seconds
```

**Return Value:**

**TRUE** if the command has been added to the transmit buffer for this UnitID  
**FALSE** if transmit buffer for this UnitID is full

**See Also:**

**ErrorBeepLED()**

**NewField** ( *UnitID* As Long, *StartPos* As Integer, *length* As Integer, *style* As Integer  
[, *Barcode* As String = ""]) As Boolean

This method defines an input field on the form, *StartPos* and *length* define the position and size of the field and the *style* is used to define the behaviour of the field, such as, how data is displayed and the field responds to key presses

**Parameters:**

<i>UnitID</i>	HHT serial number		
<i>StartPos</i>	Position of the input field		
<i>length</i>	Field length		
<i>style</i>	Value	SYNTAX	DESCRIPTION
	<b>&amp;h01</b>	SND_ENTER	Field is sent to host by pressing the ENT key
	<b>&amp;h02</b>	NO_SEND	Field is not sent if the SND_ALL command occurs. Field will be sent, if the field itself gave the SND_ALL command
	<b>&amp;h04</b>	SND_ALL	All fields in the page, (except NO_SEND fields) are sent to the host when the ENT key is pressed
	<b>&amp;h08</b>	FLD_LOCK	Field is locked. Field cannot be written to (used for "button" style fields)
	<b>&amp;h10</b>	FLD_LINE	Field is underlined
	<b>&amp;h20</b>	FLD_READER	Field can be filled with laser or external scanner data
	<b>&amp;h40</b>	FLD_CLR	Field is cleared if ACK is received
	<b>&amp;h80</b>	FLD_ACTIVE	Field is active (it has a cursor) when the form is first loaded
	<b>&amp;h100</b>	FLD_LARGE	Text in the field is double height
<b>&amp;h200</b>	FLD_PASS	Password field, text entered appears as *****	
<i>barcode</i>	If the field is FLD_READER then by default the scanner will read all code types. Use this field to restrict the reader to a particular code and length. Please refer to appendix A for more details.		

SND\_ENTER and FLD\_READER combinations:

SND_ENTER	FLD_READER	FUNCTION
0	0	Field data is not sent by pressing the ENT key, and the scanner doesn't fill the field
0	1	Field data is not sent by pressing the ENT key, but the scanner does fill the field with data if it is the active field
1	0	Field data is sent by pressing the ENT key, but the scanner doesn't fill the field
1	1	Field data is sent by pressing the ENT key or by scanner filling the field with data

**Example:**

```
Const SND_ENTER As Byte = 1 'Setup style constants
Const NO_SEND As Byte = 2
Const SND_ALL As Byte = 4
Const FLD_LOCK As Byte = 8
Const FLD_LINE As Byte = 16
Const FLD_READER As Byte = 32
Const FLD_CLR As Byte = 64
Const FLD_ACTIVE As Byte = 128
Const FLD_LARGE As Byte = 256
Const FLD_PASS As Byte = 512
```

```
NewField(unitid, 20, 10, SND_ENTER + SND_ALL + FLD_READER +
FLD_ACTIVE)
'Creates a new field at position 20 which is 10 characters long, data
is sent when the ENT key is pressed, all data on the form is sent,
field can be filled by the scanner and field is active when the form
loads
```

```
NewField(unitid, 40, 10, SND_ENTER + FLD_ACTIVE + FLD_PASS)
'Creates a new field at position 40 which is 10 characters long, data
is sent when the ENT key is pressed, the field is active when the
form loads and the field is a password field.
```

A maximum of 20 input fields can be used on one form.

**Return Value:**

**TRUE** if the command has been added to the transmit buffer for this UnitID  
**FALSE** if transmit buffer for this UnitID is full

**See Also:**

**NewFieldEx()**

**NewFieldEx** ( *UnitID* As Long, *StartPos* As Integer, *length* As Integer, *style* As Integer  
[, *Barcode* As String = ""]) As Boolean

Defines a new input field. This method is similar to the **NewField()** method, the difference being the style bit's OVR and READER\_DEFAULT replace bit's FLD\_LINE and FLD\_CLR.

As the FLD\_LINE bit no longer exists the field is underlined automatically.

**Parameters:**

<i>UnitID</i>	HHT serial number		
<i>StartPos</i>	Position of the input field		
<i>length</i>	Field length		
<i>style</i>	Value	SYNTAX	DESCRIPTION
	<b>&amp;h01</b>	SND_ENTER	Field is sent to host by pressing the ENT key
	<b>&amp;h02</b>	NO_SEND	Field is not sent if the SND_ALL command occurs. Field will be sent, if the field itself gave the SND_ALL command
	<b>&amp;h04</b>	SND_ALL	All fields in the page, (except NO_SEND fields) are sent to the host when the ENT key is pressed
	<b>&amp;h10</b>	OVR	Overwrite mode. When OVR is set and the field becomes active, the cursor moves to the starting position of the field. When the cursor is in the starting position, the previous text will be overwritten by typing
	<b>&amp;h20</b>	FLD_READER	Field can be filled with laser or external scanner data
	<b>&amp;h40</b>	READER_DEFAULT	If another field is active and it doesn't have FLD_READER set, the scanner is activated and data goes to this field automatically. Only one READER_DEFAULT field can be defined per form. This bit has no effect if FLD_READER bit is not set
	<b>&amp;h80</b>	FLD_ACTIVE	Field is active (it has a cursor) when the form is first loaded
	<b>&amp;h100</b>	FLD_LARGE	Text in the field is double height
	<b>&amp;h200</b>	FLD_PASS	Password field, text entered appears as *****
<i>barcode</i>	If the field is FLD_READER then by default the scanner will read all code types. Use this field to restrict the reader to a particular code and length. Please refer to appendix A for more details. NOTE. If a field on the form is declared as READER_DEFAULT then the barcode settings for that field will be the default for all fields on the form.		

SND\_ENTER and FLD\_READER combinations are the same as **NewField()**.

**Example:**

```
Const SND_ENTER As Byte = 1 'Setup style constants
Const NO_SEND As Byte = 2
Const SND_ALL As Byte = 4
Const OVR As Byte = 16
Const FLD_READER As Byte = 32
Const READER_DEFAULT As Byte = 64
Const FLD_ACTIVE As Byte = 128
Const FLD_LARGE As Byte = 256
Const FLD_PASS As Byte = 512
```

```
NewField(unitid, 20, 10, SND_ENTER + OVR + FLD_READER +
READER_DEFAULT + FLD_ACTIVE)
```

'Creates a new field at position 20 which is 10 characters long, data is sent when the ENT key is pressed, when the field gets focus the cursor is in position to overwrite data, field can be filled by the scanner, any data from the scanner goes to this field and the field is active when the form loads

A maximum of 20 input fields can be used on one form.

**Return Value:**

**TRUE** if the command has been added to the transmit buffer for this UnitID  
**FALSE** if transmit buffer for this UnitID is full

**See Also:**

**NewField()**

---

**PopMessage** ( *UnitID* As Long, *StartPos* As Integer, *LineText* As String [, *ReverseText* As Boolean = False] [, *LargeText* As Boolean = False]) As Boolean

This method displays a message before the form is loaded that remains on the screen until a key is pressed. Only one PopMessage can be defined per form. The *ReverseText* and *LargeText* parameters allow for different styles of message to be displayed.

**Parameters:**

<i>UnitID</i>	HHT serial number
<i>StartPos</i>	Start position of the message (0 - 79)
<i>LineText</i>	Text for the field
<i>ReverseText</i> (Optional)	Reverse text colours, white text on black background
<i>LargeText</i> (Optional)	Double height text

**Return Value:**

**TRUE** if the command has been added to the transmit buffer for this UnitID  
**FALSE** if transmit buffer for this UnitID is full

---

**Send** ( *UnitID* As Long, *FrameID* As Integer ) As Boolean

This method sends the current form to the handheld terminal.

**Parameters:**

<i>UnitID</i>	HHT serial number
<i>FrameID</i>	The number of form defined by the user. If frameid is -1, the previously defined frameid is used

**Return Value:**

**TRUE** if the command has been added to the transmit buffer for this UnitID  
**FALSE** if transmit buffer for this UnitID is full

---

**SetFormID** ( *UnitID* As Long, *FormID* As Integer ) As Boolean

This binds a specific ID number to the form, this is useful if the application uses many different forms. It allows upto 65535 different form IDs to be defined. The *formID* is sent by the handheld terminal every time data is sent, this allows the application to process the data differently depending on the form ID that is sent with it. If the *formID* is 0 then the form ID is not sent back to the host. An alternative to using form IDs is using frame IDs, these are defined in the **Send()** method, the main advantage of using form IDs over frame IDs is that when a function key is press the unit sends the frame ID as -1, however the form ID will be whatever it was set to using the **SetFormID()**. **SetFormID()** should be called just before the **Send()** method.

**Parameters:**

<i>UnitID</i>	HHT serial number
<i>FormID</i>	User defined value 0 - 65535. 0 = not used

**Return Value:**

**TRUE** if the command has been added to the transmit buffer for this UnitID  
**FALSE** if transmit buffer for this UnitID is full

**See Also:**

**GetFormID()**

---

**Text** ( *UnitID* As Long, *StartPos* As Integer, *LineText* As String [, *ReverseText* As Boolean = False] [, *LargeText* As Boolean = False] ) As Boolean

The **Text()** method displays a text string at a specific position on the form. The style of the text can be changed using the *ReverseText* and *LargeText* parameters.

**Parameters:**

<i>UnitID</i>	HHT serial number
<i>StartPos</i>	Start position of the message (0 - 79)
<i>LineText</i>	Text for the field
<i>ReverseText</i> (Optional)	Reverse text colours, white text on black background
<i>LargeText</i> (Optional)	Double height text

**Return Value:**

**TRUE** if the command has been added to the transmit buffer for this UnitID  
**FALSE** if transmit buffer for this UnitID is full

---

**WaitSerial** ( *UnitID* As Long, *Duration* As Integer ) As Boolean

Allows data in from the serial port of the unit. If the duration is between 1 and 99, then the serial port is open for that amount of time in seconds, any data that comes in from the serial port at that time is used to fill the currently active field on the form. If the duration is set to 255 then the serial port is opened for 30 seconds by default and the data from the serial port is sent directly to the host application, where the data can be accessed using the **GetSerialData()** method. **Serial settings are fixed to: 19200,n,8,1**

**Parameters:**

<i>UnitID</i>	HHT serial number
<i>Duration</i>	Time to keep the serial port open (1-99) seconds

**Example:**

**Using the duration variable:**

```
Sub CreateForm() 'This routine defines the forms objects
...
RFeasyServ.NewField(UnitID, 0, 20, FLD_ACTIVE) 'Creates the active
field for the data to go into
RFeasyServ.WaitSerial(UnitID, 20) 'Opens the serial port for 20
seconds
RFeasyServ.Send(UnitID, FrameID)
...
End Sub
```

The data from the serial port goes into the active field.

**Using the GetSerialData() method:**

```
Sub CreateForm() 'This routine defines the forms objects
...
RFeasyServ.WaitSerial(UnitID, 255) 'Opens the serial port using the
duration value 255
RFeasyServ.Send(UnitID, FrameID)
...
End Sub
Sub ProcessForm() 'This routine processes data from the form
...
mydata = RFeasyServ.GetSerialData(UnitID) 'Gets the serial data
...
End Sub
```

Data from the serial port goes directly back to the host application where the data is processed

**Return Value:**

**TRUE** if the command has been added to the transmit buffer for this UnitID  
**FALSE** if transmit buffer for this UnitID is full

**See Also:**

**GetSerialData()**

---

**What** ( *UnitID* As Long, *Delay* As Integer ) As Boolean

After the *Delay* time has expired the unit sends a What command back to the host application, this raises an event with the host application, which can then send more data to the unit.

One use for this is sending large amounts of data to the unit's serial port, by splitting up the data into smaller pieces then sending the next piece every time the What event is raised until all the data is sent.

**Parameters:**

<i>UnitID</i>	HHT serial number
<i>Delay</i>	Delay time 1-99 seconds

**Example:**

**Printing large messages to the serial port:**

```
Sub CreateForm() 'This routine defines the forms objects
...
counter = 1
RFeasyServ.DataToSerial(UnitID, string(counter)) 'Send the first
piece of data to the serial port
RFeasyServ.What(UnitID, 1) 'Creates a What event every second
RFeasyServ.Send(UnitID, FrameID)
...
End Sub
Sub RFeasyServ_WhatEv(UnitID As Long) 'This routine handles the What
event
...
counter = counter + 1
RFeasyServ.DataToSerial(UnitID, string(counter)) 'Send the next piece
of data to the serial port
RFeasyServ.What(UnitID, 1) 'Continues the What event every second
RFeasyServ.Send(UnitID, -1) 'FrameID -1 means same as before
...
End Sub
```

**Return Value:**

**TRUE** if the command has been added to the transmit buffer for this UnitID  
**FALSE** if transmit buffer for this UnitID is full

**Note:**

The **WaitSerial()** method cannot be used same time as this method.

# EVENTS

---

## **DataArrived** ( *UnitID* As Long, *FrameID* As Integer )

This event is raised when data is received from the unit, the host processes the data received and then sends a response to the unit.

Only one **DataArrived()** can be launched at anytime. If the host application is only dealing with a few handhelds then this should not cause a problem, however if a lot of units are being used then it is recommended that a separate thread is created each time the **DataArrived()** event is raised. This allows the application to process requests from many handhelds at the same time.

### Parameters:

<i>UnitID</i>	HHT serial number
<i>FrameID</i>	The number of the form as defined by the user. If the <i>frameid</i> is -1, the data comes from the initial screen or from a function key. This value is defined in <b>Send()</b> method

### See Also:

**Send( )**  
**GetData( )**  
**IsData( )**

---

## **WhatEv** ( *UnitID* As Long )

The **WhatEv()** event is raised when the handheld unit sends back a What command. The What command is sent when the *Delay* time, specified in the **What()** method, expires.

### Parameters:

<i>UnitID</i>	HHT serial number
---------------	-------------------

### See Also:

**What()**

# Appendix A

---

## Barcode type

When creating data entry fields using the NewField & NewFieldEx commands you can restrict the barcode reader to a certain symbology & code length. This is useful if you have a form with multiple fields and want to ensure the correct code is scanned at each one.

### Symbology Restriction

To restrict the barcode types specify the following in the barcode field.

"A" = EAN & UPC  
"I" = ITF (Interleaved 2of5)  
"H" = STF (Standard 2of5)  
"N" = Codabar (NW-7) "  
"M" = Code 39  
"L" = Code 93  
"K" = Code 128 (& EAN 128)

### EAN Options

To restrict the EAN codes add the following.

"A" - EAN-13 or UPC-A  
"B" - EAN-8  
"C" - UPC-E

To restrict the country codes add the 1<sup>st</sup> and 2<sup>nd</sup> character, or add "?" for a wildcard.

e.g. "A:A50" for EAN-13 & UPC-A codes starting "50"

### Other Options

To restrict the length on other codes add the min and max lengths. If these are the same or only the min is declared then only codes of that length will be read. This can be applied all types except EAN. e.g.

"I:10"            for 10 digit ITF  
"M:12-14"       for 12 to 14 digit Code 39

Some symbologies have optional Check Digits, to force the reader to validate the check digit. Add a "C" to the codes I H & M. e.g.

"I:10-14C"      for 10 to 14 digit ITF with valid Check Digit

### Examples

"A"                All EAN & UPC only  
"A:B49"          EAN 8 only starting "49"  
"I:10"            10 digit ITF only  
"K:6-12"         6 to 12 digit Code 128 only  
"M:8-12C"       8 to 12 digit Code 39 with Check digit only