MSCOMM – Visual Basic [Serial Port Functions]

Prepared by: Dr. Saeed. R. Taghizadeh [Source: Microsoft MSDN]

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/comm98/dt_vbobjComm_P.asp

Break Property

See Also Example Applies To

Sets or clears the break signal state. This property is not available at design time.

Syntax

object.**Break** [= *value*]

The **Break** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	A Boolean expression specifying whether the break signal state is set, as described in Settings.

Settings

The settings for *value* are:

Setting	Description
True	Sets the break signal state.
False	Clears the break signal state.

Remarks

When set to **True**, the **Break** property sends a break signal. The break signal suspends character transmission and places the transmission line in a break state until you set the **Break** property to **False**. Typically, you set the break state for a short interval of time, and *only* if the device with which you are communicating requires that a break signal be set.

Data Type

Boolean

Break Property Example

The following example shows how to send a break signal for a tenth of a second:

```
' Set the Break condition.
MSComm1.Break = True
' Set duration to 1/10 second.
Duration! = Timer + .1
' Wait for the duration to pass.
Do Until Timer > Duration!
   Dummy = DoEvents()
Loop
' Clear the Break condition.
MSComm1.Break = False
```

CDHolding Property

See Also Example Applies To

Determines whether the carrier is present by querying the state of the Carrier Detect (CD) line. Carrier Detect is a signal sent from a modem to the attached computer to indicate that the modem is online. This property is not available at design time and is read-only at run time.

Syntax

object.**CDHolding**

The **CDHolding** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Settings

The settings for the **CDHolding** property are:

Setting	Description
True	Carrier Detect line is high
False	Carrier Detect line is low

Remarks

Note It is especially important to trap a loss of the carrier in a host application, such as a bulletin board, because the caller can hang up (drop the carrier) at any time.

The Carrier Detect is also known as the Receive Line Signal Detect (RLSD).

Data Type

Boolean

CTSHolding Property

See Also Example Applies To

Determines whether you can send data by querying the state of the Clear To Send (CTS) line. Typically, the Clear To Send signal is sent from a modem to the attached computer to indicate that transmission can proceed. This property is not available at design time and is read-only at run time.

Syntax

object.**CTSHolding**

The **CTSHolding** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

The following table lists the **CTSHolding** property settings for the **MSComm** control.

Setting	Description
True	Clear To Send line high.
False	Clear To Send line low.

Remarks

The Clear To Send line is used in RTS/CTS (Request To Send/Clear To Send) hardware handshaking. The **CTSHolding** property gives you a way to manually poll the Clear To Send line if you need to determine its state.

For more information on handshaking protocols, see the Handshaking property.

Data Type

Boolean

CommEvent Property

See Also Example Applies To

Returns the most recent communication event or error. This property is not available at design time and is read-only at run time.

Syntax

object.**CommEvent**

The **CommEvent** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Remarks

Although the **OnComm** event is generated whenever a communication error or event occurs, the **CommEvent** property holds the numeric code for that error or event. To determine the actual error or event that caused the **OnComm** event, you must reference the **CommEvent** property.

The **CommEvent** property returns one of the following values for communication errors or events. These constants can also be found in the Object Library for this control.

Communication errors include the following settings:

Constant	Value	Description
comEventBreak	1001	A Break signal was received.
comEventFrame	1004	Framing Error. The hardware detected a framing error.
comEventOverrun	1006	Port Overrun. A character was not read from the hardware before the next character arrived and was lost.
comEventRxOver	1008	Receive Buffer Overflow. There is no room in the receive buffer.
comEventRxParity	1009	Parity Error. The hardware detected a parity error.
comEventTxFull	1010	Transmit Buffer Full. The transmit buffer was full while trying to queue a character.
comEventDCB	1011	Unexpected error retrieving Device Control Block (DCB) for the port.

Communications events include the following settings:

Constant	Value	Description
comEvSend	1	There are fewer than Sthreshold number of characters in the transmit buffer.
comEvReceive	2	Received Rthreshold number of characters. This event is generated continuously until you use the Input property to remove the data from the receive buffer.
comEvCTS	3	Change in Clear To Send line.
comEvDSR	4	Change in Data Set Ready line. This event is only fired when DSR changes from 1 to 0.
comEvCD	5	Change in Carrier Detect line.
comEvRing	6	Ring detected. Some UARTs (universal asynchronous receiver-transmitters) may not support this event.
comEvEOF	7	End Of File (ASCII character 26) character received.

Data Type

Integer

CommID Property

See Also Example Applies To

Returns a handle that identifies the communications device. This property is not available at design time and is read-only at run time.

Syntax

object.**CommID**

The **CommID** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Remarks

This is the same value that's returned by the Windows API **CreateFile** function. Use this value when calling any communications routines in the Windows API.

Data Type

Long

CommPort Property

See Also Example Applies To

Sets and returns the communications port number.

Syntax

object.**CommPort**[= *value*]

The **CommPort** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	A integer value specifying the port number.

Remarks

You can set *value* to any number between 1 and 16 at design time (the default is 1). However, the **MSComm** control generates error 68 (`Device unavailable`) if the port does not exist when you attempt to open it with the **PortOpen** property.

Warning You must set the **CommPort** property before opening the port.

Data Type

Integer

DSRHolding Property

See Also Example Applies To

Determines the state of the Data Set Ready (DSR) line. Typically, the Data Set Ready signal is sent by a modem to its attached computer to indicate that it is ready to operate. This property is not available at design time and is read-only at run time.

Syntax

object.**DSRHolding**

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list.

The **DSRHolding** property returns the following values:

Value	Description
True	Data Set Ready line high
False	Data Set Ready line low

Remarks

This property is useful when writing a Data Set Ready/Data Terminal Ready handshaking routine for a Data Terminal Equipment (DTE) machine.

Data Type

Boolean

DTREnable Property

See Also Example Applies To

Determines whether to enable the Data Terminal Ready (DTR) line during communications. Typically, the Data Terminal Ready signal is sent by a computer to its modem to indicate that the computer is ready to accept incoming transmission.

Syntax

object.**DTREnable**[= *value*]

The **DTREnable** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
Value	A Boolean expression specifying whether to enable the Data Terminal Ready (DTR) line, as described in Settings.

Settings

The settings for *value* are:

Setting	Description
True	Enable the Data Terminal Ready line.
False	(Default) Disable the Data Terminal Ready line.

Remarks

When **DTREnable** is set to **True**, the Data Terminal Ready line is set to high (on) when the port is opened, and low (off) when the port is closed. When **DTREnable** is set to **False**, the Data Terminal Ready always remains low.

Note In most cases, setting the Data Terminal Ready line to low hangs up the telephone.

Data Type

Boolean

EOFEnable Property

See Also Example Applies To

The **EOFEnable** property determines if the **MSComm** control looks for End Of File (EOF) characters during input. If an EOF character is found, the input will stop and the **OnComm** event will fire with the **CommEvent** property set to **comEvEOF**.

Syntax

object.**EOFEnable** [= *value*]

The **EOFEnable** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	A boolean expression that determines whether the OnComm event is fired when an EOF character is found, as described in Settings.

Settings

The settings for *value* are:

Setting	Description
True	The OnComm event is fired when an EOF character is found.
False	(Default) The OnComm event isn't fired when an EOF character is found.

Remarks

When **EOFEnable** property is set to **False**, the control will not scan the input stream for EOF characters.

Handshaking Property

See Also Example Applies To
Sets and returns the hardware handshaking protocol.
Syntax
object.**Handshaking** [= *value*]
The **Handshaking** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	An integer expression specifying the handshaking protocol, as described in Settings.

Settings
The settings for *value* are:

Setting	Value	Description
comNone	0	(Default) No handshaking.
comXOnXOff	1	XON/XOFF handshaking.
comRTS	2	RTS/CTS (Request To Send/Clear To Send) handshaking.
comRTSXOnXOff	3	Both Request To Send and XON/XOFF handshaking.

Remarks
Handshaking refers to the internal communications protocol by which data is transferred from the hardware port to the receive buffer. When a character of data arrives at the serial port, the communications device has to move it into the receive buffer so that your program can read it. If there is no receive buffer and your program is expected to read every character directly from the hardware, you will probably lose data because the characters can arrive very quickly.
A **handshaking** protocol insures data is not lost due to a buffer overrun, where data arrives at the port too quickly for the communications device to move the data into the receive buffer.
Data Type
Integer

InBufferCount Property

See Also Example Applies To
Returns the number of characters waiting in the receive buffer. This property is not available at design time.
Syntax
object.**InBufferCount**[= *value*]
The **InBufferCount** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	An integer expression specifying the number of characters waiting in the receive buffer.

Remarks
InBufferCount refers to the number of characters that have been received by the modem and are waiting in the receive buffer for you to take them out. You can clear the receive buffer by setting the **InBufferCount** property to 0.
Note Do not confuse this property with the **InBufferSize** property. The **InBufferSize** property reflects the total size of the receive buffer.
Data Type
Integer

InBufferSize Property

See Also Example Applies To
Sets and returns the size of the receive buffer in bytes.

Syntax
object.**InBufferSize**[= *value*]
The **InBufferSize** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	An integer expression specifying the size of the receive buffer in bytes.

Remarks
InBufferSize refers to the total size of the receive buffer. The default size is 1024 bytes. Do not confuse this property with the **InBufferCount** property which reflects the number of characters currently waiting in the receive buffer.
Note Note that the larger you make the receive buffer, the less memory you have available to your application. However, if your buffer is too small, it runs the risk of overflowing unless handshaking is used. As a general rule, start with a buffer size of 1024 bytes. If an overflow error occurs, increase the buffer size to handle your application's transmission rate.
Data Type
Integer

Index Property (ActiveX Controls)

See Also Example Applies To
Returns or sets the number that uniquely identifies an object in a collection.
Syntax
object.**Index**
The object placeholder is an object expression that evaluates to an object in the Applies To list.
Remarks
The **Index** property is set by default to the order of the creation of objects in a collection. The index for the first object in a collection will always be one (1).
The value of the **Index** property of an object can change when objects in the collection are reordered, such as when you set the **Sorted** property to **True**. If you expect the **Index** property to change dynamically, it may be more useful to refer to objects in a collection by using the **Key** property.

Button Object
ColumnHeader Object, ColumnHeaders Collection
ListImage Object, ListImages Collection
ListItem Object, ListItems Collection
Node Object, Nodes Collection
Panel Object
SSTab Control
Tab Object

Button Object

See Also Example Properties Methods Events
A **Button** object represents an individual button in the **Buttons** collection of a **Toolbar** control.
Remarks
For each **Button** object, you can add text or a bitmap image, or both, from an **ImageList** control, and set properties to change its state and style.
At design time, use the Insert Button and Remove Button buttons on the Buttons tab in the Properties Page of the **Toolbar** control to insert and remove **Button** objects from the **Buttons** collection. At run time, you can also add **Button** objects by using the **Add** method of the **Buttons** collection.
At design time and run time, you can set the **Caption**, **Image**, **Value**, **MixedState**, and **ToolTipText** properties to change the appearance of each **Button** object.

Whenever a button is clicked on the **Toolbar** control, the ButtonClick event is called with the selected **Button** object passed in as a parameter. To cause some action to occur when a button is clicked, use the **Index** or **Key** properties in a **Select Case** statement as in the following code:

```
Select Case Button.Key
  Case Is = "open" ' Open file.
  ' Add code to Open a file here
  Case Is = "save" ' Save file.
  ' Add code to Save a file here
  Case Else
  ' If any other button is pressed
End Select
```

ColumnHeader Object, ColumnHeaders Collection

See Also Example Properties Methods Events

- A **ColumnHeader** object is an item in a **ListView** control that contains heading text.

- A **ColumnHeaders** collection contains one or more **ColumnHeader** objects.

Syntax

listview.**ColumnHeaders**
listview.**ColumnHeaders**(*index*)

The syntax lines above refer to the collection and to individual elements in the collection, respectively, according to the standard collection syntax.

The **ColumnHeader** object, **ColumnHeaders** collection syntax has these parts:

Part	Description
listview	An object expression that evaluates to a **ListView** control.
index	Either an integer or string that uniquely identifies a member of an object collection. An integer would be the value of the **Index** property; a string would be the value of the **Key** property.

Remarks

You can view **ColumnHeader** objects in Report view only.

You can add **ColumnHeader** objects to a **ListView** control at both design time and run time.

With a **ColumnHeader** object, a user can:

- Click it to trigger the **ColumnClick** event and sort the items based on that data item.

- Grab the object's right border and drag it to adjust the width of the column.

- Hide **ColumnHeader** objects in Report view.

There is always one column in the **ListView** control, which is Column 1. This column contains the actual **ListItem** objects; not their subitems. The second column (Column 2) contains subitems. Therefore, you always have one more **ColumnHeader** object than subitems and the **ListItem** object's **SubItems** property is a 1-based array of size `ColumnHeaders.Count - 1`.

The number of **ColumnHeader** objects determines the number of subitems each **ListItem** object in the control can have. When you delete a **ColumnHeader** object, all of the subitems associated with the column are also deleted, and each **ListItem** object's subitem array shifts to update the indices of the **ColumnHeader**, causing the remaining column headers' **SubItemIndex** properties to change

Input Property

See Also Example Applies To

Returns and removes a stream of data from the receive buffer. This property is not available at design time and is read-only at run time.

Syntax

object.**Input**

The **Input** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

Remarks

The **InputLen** property determines the number of characters that are read by the **Input** property. Setting **InputLen** to 0 causes the **Input** property to read the entire contents of the receive buffer.

The **InputMode** property determines the type of data that is retrieved with the **Input** property. If **InputMode** is set to **comInputModeText** then the **Input** property returns text data in a **Variant**. If **InputMode** is **comInputModeBinary** then the **Input** property returns binary data in an array of bytes in a **Variant**.

Data Type

Variant

Input Property Example

This example shows how to retrieve data from the receive buffer:

```
Private Sub Command1_Click()
Dim InString as String
' Retrieve all available data.
MSComm1.InputLen = 0

' Check for data.
If MSComm1.InBufferCount Then
   ' Read data.
   InString = MSComm1.Input
End If
End Sub
```

InputLen Property

See Also Example Applies To

Sets and returns the number of characters the **Input** property reads from the receive buffer.

Syntax

object.**InputLen** [= *value*]

The **InputLen** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	An integer expression specifying the number of characters the **Input** property reads from the receive buffer.

Remarks

The default value for the **InputLen** property is 0. Setting **InputLen** to 0 causes the **MSComm** control to read the entire contents of the receive buffer when **Input** is used.

If **InputLen** characters are not available in the receive buffer, the **Input** property returns a zero-length string (""). The user can optionally check the **InBufferCount** property to determine if the required number of characters are present before using **Input**.

This property is useful when reading data from a machine whose output is formatted in fixed-length blocks of data.
Data Type
Integer

InputLen Property Example

This example shows how to read 10 characters of data:

```
Private Command1_Click()
Dim CommData as String
' Specify a 10 character block of data.
MSComm1.InputLen = 10
' Read data.
CommData = MSComm1.Input
End Sub
```

InputMode Property

See Also Example Applies To
Sets or returns the type of data retrieved by the **Input** property.
Syntax
object.**InputMode** [= *value*]
The **InputMode** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	A value or constant that specifies the input mode, as described in Settings.

Settings
The settings for *value* are:

Constant	Value	Description
comInputModeText	0	(Default) Data is retrieved through the **Input** property as text.
comInputModeBinary	1	Data is retrieved through the **Input** property as binary data.

Remarks
The **InputMode** property determines how data will be retrieved through the **Input** property. The data will either be retrieved as string or as binary data in a byte array.
Use **comInputModeText** for data that uses the ANSI character set. Use **comInputModeBinary** for all other data such as data that has embedded control characters, Nulls, etc.

InputMode Property Example

This example reads 10 bytes of binary data from the communications port and assigns it to a byte array.

```
Private Sub Command1_Click()
Dim Buffer as Variant
Dim Arr() as Byte

' Set and open port
MSComm1.CommPort = 1
MSComm1.PortOpen = True

' Set InputMode to read binary data
MSComm1.InputMode = comInputModeBinary
```

```
' Wait until 10 bytes are in the input buffer
Do Until MSComm1.InBufferCount < 10
  DoEvents
Loop

' Assign to byte array for processing
Arr = MSComm1.Input

End Sub
```

Name Property

See Also Example Applies To

- Returns the name used in code to identify a form, control, or data access object. Read-only at run time.

- Returns or sets the name of a font object.

Syntax

object.**Name**

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list. If *object* is omitted, the form associated with the active form module is assumed to be *object.*

Remarks

The default name for new objects is the kind of object plus a unique integer. For example, the first new **Form** object is Form1, a new **MDIForm** object is MDIForm1, and the third **TextBox** control you create on a form is Text3.

An object's **Name** property must start with a letter and can be a maximum of 40 characters. It can include numbers and underline (_) characters but can't include punctuation or spaces. Forms can't have the same name as another public object such as **Clipboard**, **Screen**, or **App**. Although the **Name** property setting can be a keyword, property name, or the name of another object, this can create conflicts in your code.

You can use a form's **Name** property with the **Dim** statement at run time to create other instances of the form. You can't have two forms with the same name at design time.

You can create an array of controls of the same type by setting the **Name** property to the same value. For example, when you set the name of all option buttons in a group to MyOpt, Visual Basic assigns unique values to the **Index** property of each control to distinguish it from others in the array. Two controls of different types can't share the same name.

Note Although Visual Basic often uses the **Name** property setting as the default value for the **Caption**, **LinkTopic**, and **Text** properties, changing one of these properties doesn't affect the others.

Changing the case of the **Name** property value for a Form or other module without otherwise changing the name itself, however, can cause a “Conflicting names” error message the next time the project containing the form or module is loaded. For example, changing “Form1” to “form1” will cause the error; changing “Form1” to “formX” will not.

The error is caused by the way module names are stored within the project file – the procedure for changing names within the project file isn’t case sensitive, while the procedure for reading names on project load is.

NullDiscard Property

See Also Example Applies To

Determines whether null characters are transferred from the port to the receive buffer.

Syntax

object.**NullDiscard** [= *value*]

The **NullDiscard** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.

value	An boolean expression specifying whether null characters are transferred from the port to the receive buffer, as described in Settings

Settings
The settings for *value* are:

Setting	Description
True	Null characters are *not* transferred from the port to the receive buffer.
False	(Default) Null characters are transferred from the port to the receive buffer.

Remarks
A null character is defined as ASCII character 0, `Chr$(0)`.
Data Type
Boolean

Object Property (ActiveX Controls)

See Also Example Applies To
Returns the object and/or a setting of an object's method or property.
Syntax
object.**Object**[.*property* | .*method*]
The **Object** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
property	Property that the object supports.
method	Method that the object supports.

Remarks
Use this property to specify an object you want to use in an Automation task.
You use the object returned by the **Object** property in an Automation task by using the properties and methods of that object. For information on which properties and methods an object supports, see the documentation for the application that created the object.

OutBufferSize Property

See Also Example Applies To
Sets and returns the size, in bytes, of the transmit buffer.
Syntax
object.**OutBufferSize** [= *object*]
The **OutBufferSize** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	An integer expression specifying the size of the transmit buffer.

Remarks
OutBufferSize refers to the total size of the transmit buffer. The default size is 512 bytes. Do not confuse this property with the **OutBufferCount** which reflects the number of bytes currently waiting in the transmit buffer.
Note The larger you make the transmit buffer, the less memory you have available to your application. However, if your buffer is too small, you run the risk of overflowing unless you use handshaking. As a general rule, start with a buffer size of 512 bytes. If an overflow error occurs, increase the buffer size to handle your application's transmission rate.
Data Type
Integer

OutBufferCount Property

See Also Example Applies To

Returns the number of characters waiting in the transmit buffer. You can also use it to clear the transmit buffer. This property is not available at design time.

Syntax

object.**OutBufferCount** [= *value*]

The **OutBufferCount** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	An integer expression specifying the number of characters waiting in the transmit buffer.

Remarks

You can clear the transmit buffer by setting the **OutBufferCount** property to 0.

Note Do not confuse the **OutBufferCount** property with the **OutBufferSize** property which reflects the total size of the transmit buffer.

Data Type

Integer

Output Property

See Also Example Applies To

Writes a stream of data to the transmit buffer. This property is not available at design time and is write-only at run time.

Syntax

object.**Output** [= *value*]

The **Output** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	A string of characters to write to the transmit buffer.

Remarks

The **Output** property can transmit text data or binary data. To send text data using the **Output** property, you must specify a **Variant** that contains a string. To send binary data, you must pass a **Variant** which contains a byte array to the **Output** property.

Normally, if you are sending an ANSI string to an application, you can send it as text data. If you have data that contains embedded control characters, Null characters, etc., then you will want to pass it as binary data.

Data Type

Variant

Output Property Example

The following example shows how to send every character the user types to the serial port:

```
Private Sub Form_KeyPress (KeyAscii As Integer)
   Dim Buffer as Variant

   ' Set and open port
   MSComm1.CommPort = 1
   MSComm1.PortOpen = True

   Buffer = Chr$(KeyAscii)
   MSComm1.Output = Buffer
End Sub
```

Parent Property

See Also Example Applies To

Returns the form, object, or collection that contains a control or another object or collection.

Syntax

object.**Parent**

The *object* placeholder represents an object expression that evaluates to an object in the Applies To list.

Remarks

Use the **Parent** property to access the properties, methods, or controls of an object's parent. For example:

```
MyButton.Parent.MousePointer = 4
```

The **Parent** property is useful in an application in which you pass objects as arguments. For example, you could pass a control variable to a general procedure in a module, and use the **Parent** property to access its parent form.

There is no relationship between the **Parent** property and the **MDIChild** property. There is, however, a parent-child relationship between an **MDIForm** object and any **Form** object that has its **MDIChild** property set to **True**.

Parent Property Example

This example passes a control from a form that doesn't have the focus to a procedure in a module, and then displays the state of the control on the parent form. To try this example, create three forms: Form1, containing a **CommandButton** control, and Form2 and Form3, each containing a **CheckBox** control. You must also create a new module (click Add Module in the Project menu). Paste the code into the Declarations sections of the respective forms or module, and then press F5 to run the program.

```
' Enter this code into Form1.
Private Sub Form_Load ()
   Form2.Show    ' Display all forms.
   Form3.Show
   Form2.AutoRedraw = True
   Form3.AutoRedraw = True
End Sub

Private Sub Command1_Click ()
   ReadCheckBox Form2.Check1    ' Call procedure in other module
   ReadCheckBox Form3.Check1    ' and send control as argument.
End Sub

' Enter this code into Module1.
Sub ReadCheckBox (Source As Control)
   If Source.Value Then
      Source.Parent.Cls    ' Clear parent form.
      Source.Parent.Print "CheckBox is ON."  ' Display on parent form.
   Else
      Source.Parent.Cls    ' Clear parent form.
      Source.Parent.Print "CheckBox is OFF."  ' Display on parent form.
   End If
End Sub
```

ParityReplace Property

See Also Example Applies To

Sets and returns the character that replaces an invalid character in the data stream when a parity error occurs.

Syntax

object.**ParityReplace** [= *value*]

The **ParityReplace** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	A string expression representing a character, as described in Remarks.

Remarks

The *parity bit* refers to a bit that is transmitted along with a specified number of data bits to provide a small amount of error checking. When you use a parity bit, the **MSComm** control adds up all the bits that are set (having a value of 1) in the data and tests the sum as being odd or even (according to the parity setting used when the port was opened).

By default, the control uses a question mark (?) character for replacing invalid characters. Setting **ParityReplace** to an empty string ("") disables replacement of the character where the parity error occurs. The **OnComm** event is still fired and the **CommEvent** property is set to **comEventRXParity**.

The **ParityReplace** character is used in a byte-oriented operation, and must be a single-byte character. You can specify any ANSI character code with a value from 0 to 255.

Data Type

String

PortOpen Property

See Also Example Applies To

Sets and returns the state of the communications port (open or closed). Not available at design time.

Syntax

object.**PortOpen** [= *value*]

The **PortOpen** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	A boolean expression specifying the state of the communications port.

Settings

The settings for *value* are:

Setting	Description
True	Port is opened
False	Port is closed

Remarks

Setting the **PortOpen** property to **True** opens the port. Setting it to **False** closes the port and clears the receive and transmit buffers. The **MSComm** control automatically closes the serial port when your application is terminated.

Make sure the **CommPort** property is set to a valid port number before opening the port. If the **CommPort** property is set to an invalid port number when you try to open the port, the **MSComm** control generates error 68 `(Device unavailable)`.

In addition, your serial port device must support the current values in the **Settings** property. If the **Settings** property contains communications settings that your hardware does not support, your hardware may not work correctly.

If either the **DTREnable** or the **RTSEnable** properties is set to **True** before the port is opened, the properties are set to **False** when the port is closed. Otherwise, the DTR and RTS lines remain in their previous state.

Data Type

Boolean

PortOpen Property Example

The following example opens communications port number 1 at 9600 baud with no parity checking, 8 data bits, and 1 stop bit:

```
MSComm1.Settings = "9600,n,8,1"
MSComm1.CommPort = 1
MSComm1.PortOpen = True
```

RTSEnable Property

See Also Example Applies To

Determines whether to enable the Request To Send (RTS) line. Typically, the Request To Send signal that requests permission to transmit data is sent from a computer to its attached modem.

Syntax

object.**RTSEnable**[= *value*]

The **RTSEnable** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	An boolean expression specifying whether the Request To Send (RTS) line is enabled, as described in Settings.

Settings

The settings for *value* are:

Setting	Description
True	Enables the Request To Send line.
False	(Default) Disables the Request To Send line.

Remarks

When **RTSEnable** is set to **True**, the Request To Send line is set to high (on) when the port is opened, and low (off) when the port is closed.

The Request To Send line is used in RTS/CTS hardware handshaking. The **RTSEnable** property allows you to manually poll the Request To Send line if you need to determine its state.

For more information on handshaking protocols, see the **Handshaking** property.

Data Type

RThreshold Property

See Also Example Applies To

Sets and returns the number of characters to receive before the **MSComm** control sets the **CommEvent** property to **comEvReceive** and generates the **OnComm** event.

Syntax

object.**Rthreshold** [= *value*]

The **Rthreshold** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	An integer expression specifying the number of characters to receive before generating the OnComm event.

Remarks

Setting the **RThreshold** property to 0 (the default) disables generating the **OnComm** event when characters are received.

Setting **RThreshold** to 1, for example, causes the **MSComm** control to generate the **OnComm** event every time a single character is placed in the receive buffer.

Data Type

Integer

SThreshold Property

See Also Example Applies To

Sets and returns the minimum number of characters allowable in the transmit buffer before the **MSComm** control sets the **CommEvent** property to **comEvSend** and generates the **OnComm** event.

Syntax

object.**SThreshold** [= *value*]

The **SThreshold** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	An integer expression representing the minimum number of characters in the transmit buffer before the OnComm event is generated.

Remarks

Setting the **SThreshold** property to 0 (the default) disables generating the **OnComm** event for data transmission events. Setting the **SThreshold** property to 1 causes the **MSComm** control to generate the **OnComm** event when the transmit buffer is completely empty.

If the number of characters in the transmit buffer is less than *value*, the **CommEvent** property is set to **comEvSend**, and the **OnComm** event is generated. The **comEvSend** event is only fired once, when the number of characters crosses the **SThreshold**. For example, if **SThreshold** equals five, the **comEvSend** event occurs only when the number of characters drops from five to four in the output queue. If there are never more than SThreshold characters in the output queue, the event is never fired.

Data Type

Integer

Settings Property

See Also Example Applies To

Sets and returns the baud rate, parity, data bit, and stop bit parameters.

Syntax

object.**Settings** [= *value*]

The **Settings** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
value	An string expression representing the communications port settings, as described below.

Remarks

If *value* is not valid when the port is opened, the **MSComm** control generates error 380 (`Invalid property value`).

Value is composed of four settings and has the following format:

`"BBBB,P,D,S"`

Where `BBBB` is the baud rate, `P` is the parity, `D` is the number of data bits, and `S` is the number of stop bits.

The default value of *value* is:

`"9600,N,8,1"`

The following table lists the valid baud rates.

Setting
110
300
600
1200
2400

4800
9600 (Default)
14400
19200
28800
38400
56000
57600
115200
128000
256000

The following table describes the valid parity values.

Setting	Description
E	Even
M	Mark
N	(Default) None
O	Odd
S	Space

The following table lists the valid data bit values.

Setting

4
5
6
7
8 (Default)

The following table lists the valid stop bit values.

Setting	
1	(Default)
1.5	
2	

Data Type
String

Settings Example

The following example sets the control's port to communicate at 9600 baud with no parity checking, 8 data bits, and 1 stop bit:

```
MSComm1.Settings = "9600,N,8,1"
```

Tag Property (ActiveX Controls)

See Also Example Applies To

Returns or sets an expression that stores any extra data needed for your program. Unlike other properties, the value of the **Tag** property isn't used by Visual Basic; you can use this property to identify objects.

Syntax
object.**Tag** [= *expression*]
The **Tag** property syntax has these parts:

Part	Description
object	An object expression that evaluates to an object in the Applies To list.
expression	A string expression identifying the object. The default is a zero-length string

	("").

Remarks
You can use this property to assign an identification string to an object without affecting any of its other property settings or causing side effects. The **Tag** property is useful when you need to check the identity of a control or **MDIForm** object that is passed as a variable to a procedure.
Tip When you create a new instance of a form, assign a unique value to the **Tag** property.
Note The **Tag** property is of type Variant for ActiveX control collections such as **Toolbar Button** objects, **TreeView Node** objects, **ListView ListItem** and **ColumnHeader** objects, **ImageList ListImage** objects, **TabStrip Tab** objects, and **StatusBar Panel** objects. You can use the **Tag** property to pass values, but it does not allow you to pass objects.

Tag Property Example

This example displays a unique icon for each control being dragged. To try this example, paste the code into the Declarations section of a form that contains three **PictureBox** controls. Set the **DragMode** property to 1 for Picture1 and Picture2, and then press F5. Use the mouse to drag Picture1 or Picture2 over Picture3 controls.

```
Private Sub Form_Load ()
  Picture1.Tag = "ICONS\ARROWS\POINT03.ICO"
  Picture2.Tag = "ICONS\ARROWS\POINT04.ICO"
End Sub

Private Sub Picture3_DragOver (Source As Control, X As Single, Y As
Single, State As Integer)
   If State = vbEnter Then
      ' Select based on each PictureBox's Name property.
      Select Case Source.Name
      Case "Picture1"
         ' Load icon for Picture1.
         Source.DragIcon = LoadPicture(Picture1.Tag)        Case
"Picture2"
         ' Load icon for Picture2.
         Source.DragIcon = LoadPicture(Picture2.Tag)
      End Select
   ElseIf State = vbLeave Then
      ' When source isn't over Picture3, unload icon.
      Source.DragIcon = LoadPicture ()
   End If
End Sub
```

```
Private Sub Form_Load()
  Left = (Screen.Width - Width) / 2
  Top = (Screen.Height - Height) / 2
  TextAddr = 768
  TextValue = 0
  OptionByte = True
  OptionHex = True
End Sub

Private Sub OnAbout_Click()
  About.Show 1
End Sub


Private Sub OnExit_Click()
  End
End Sub


Private Sub OnRead_Click()
  Dim Value As Long

  If OptionByte Then
    Value = DlPortReadPortUchar(Val(TextAddr))
  ElseIf OptionWord Then
    Value = DlPortReadPortUshort(Val(TextAddr))
    If Value < 0 Then Value = Value + 65536
  Else
    Value = DlPortReadPortUlong(Val(TextAddr))
  End If

  If OptionDec Then
    TextValue = Value
  Else
    TextValue = "&H" + Hex(Value)
  End If
End Sub

Private Sub OnWrite_Click()
  Dim Value As Long

  Value = Val(TextValue)
  If OptionByte Then
    If Value < 0 Then Value = Value + 256
    DlPortWritePortUchar Val(TextAddr), Value
  ElseIf OptionWord Then
    If Value < 0 Then Value = Value + 65536
    DlPortWritePortUshort Val(TextAddr), Value
  Else
    DlPortWritePortUlong Val(TextAddr), Value
  End If
End Sub


Private Sub OptionDec_Click()
  TextAddr = Val(TextAddr)
  TextValue = Val(TextValue)
End Sub
```

```
Private Sub OptionHex_Click()
  TextAddr = "&H" + Hex(Val(TextAddr))
  TextValue = "&H" + Hex(Val(TextValue))
End Sub


Private Sub OptionWord_Click()

End Sub







Private Sub Option3_Click()

End Sub


Private Sub Option4_Click()

End Sub


Private Sub Form_Load()
  Left = (Screen.Width - Width) / 2
  Top = (Screen.Height - Height) / 2
  TextAddr = 768
  TextValue = 0
  OptionByte = True
  OptionHex = True
End Sub

Private Sub Frame1_DragDrop(Source As Control, X As Single, Y As
Single)

End Sub

Private Sub OnAbout_Click()
  About.Show 1
End Sub


Private Sub OnExit_Click()
  End
End Sub


Private Sub OnRead_Click()
  Dim Value As Long

  If OptionByte Then
    Value = DlPortReadPortUchar(Val(TextAddr))
  ElseIf OptionWord Then
    Value = DlPortReadPortUshort(Val(TextAddr))
    If Value < 0 Then Value = Value + 65536
```

```
  Else
    Value = DlPortReadPortUlong(Val(TextAddr))
  End If

  If OptionDec Then
    TextValue = Value
  Else
    TextValue = "&H" + Hex(Value)
  End If
End Sub

Private Sub OnWrite_Click()
  Dim Value As Long

  Value = Val(TextValue)
  If OptionByte Then
    If Value < 0 Then Value = Value + 256
    DlPortWritePortUchar Val(TextAddr), Value
  ElseIf OptionWord Then
    If Value < 0 Then Value = Value + 65536
    DlPortWritePortUshort Val(TextAddr), Value
  Else
    DlPortWritePortUlong Val(TextAddr), Value
  End If
End Sub


Private Sub OptionDec_Click()
  TextAddr = Val(TextAddr)
  TextValue = Val(TextValue)
End Sub

Private Sub OptionHex_Click()
  TextAddr = "&H" + Hex(Val(TextAddr))
  TextValue = "&H" + Hex(Val(TextValue))
End Sub


Private Sub OptionWord_Click()

End Sub


Private Sub Frame2_DragDrop(Source As Control, X As Single, Y As
Single)

End Sub

Private Sub OnAbout_Click()
  About.Show 1
End Sub


Private Sub OnExit_Click()
  End
End Sub
```

```
Private Sub OnRead_Click()
  Dim Value As Long

  If OptionByte Then
    Value = DlPortReadPortUchar(Val(TextAddr))
  ElseIf OptionWord Then
    Value = DlPortReadPortUshort(Val(TextAddr))
    If Value < 0 Then Value = Value + 65536
  Else
    Value = DlPortReadPortUlong(Val(TextAddr))
  End If

  If OptionDec Then
    TextValue = Value
  Else
    TextValue = "&H" + Hex(Value)
  End If
End Sub

Private Sub OnWrite_Click()
  Dim Value As Long

  Value = Val(TextValue)
  If OptionByte Then
    If Value < 0 Then Value = Value + 256
    DlPortWritePortUchar Val(TextAddr), Value
  ElseIf OptionWord Then
    If Value < 0 Then Value = Value + 65536
    DlPortWritePortUshort Val(TextAddr), Value
  Else
    DlPortWritePortUlong Val(TextAddr), Value
  End If
End Sub


Private Sub OptionDec_Click()
  TextAddr = Val(TextAddr)
  TextValue = Val(TextValue)
End Sub

Private Sub OptionHex_Click()
  TextAddr = "&H" + Hex(Val(TextAddr))
  TextValue = "&H" + Hex(Val(TextValue))
End Sub


Private Sub OptionWord_Click()

End Sub
```

IEEE 1284 Type C however, is a 36 conductor connector like the Centronics, but smaller. This connector is claimed to have a better clip latch, better electrical properties and is easier to assemble. It also contains two more pins for signals which can be used to see whether the other device connected, has power. 1284 Type C connectors are recommended for new designs, so we can look forward on seeing these new connectors in the near future.

Pin No (D-Type 25)	Pin No (Centronics)	SPP Signal	Direction In/out	Register	Hardware Inverted
1	1	nStrobe	In/Out	Control	Yes
2	2	Data 0	Out	Data	
3	3	Data 1	Out	Data	
4	4	Data 2	Out	Data	
5	5	Data 3	Out	Data	
6	6	Data 4	Out	Data	
7	7	Data 5	Out	Data	
8	8	Data 6	Out	Data	
9	9	Data 7	Out	Data	
10	10	nAck	In	Status	
11	11	Busy	In	Status	Yes
12	12	Paper-Out / Paper-End	In	Status	
13	13	Select	In	Status	
14	14	nAuto-Linefeed	In/Out	Control	Yes
15	32	nError / nFault	In	Status	
16	31	nInitialize	In/Out	Control	
17	36	nSelect-Printer / nSelect-In	In/Out	Control	Yes
18 - 25	19-30	Ground	Gnd		