Winsock TCP Control

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vbobjWinsockControlC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vbobjWinsockControlX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"vbobjWinsockControlP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"vbobjWinsockControlM"} {ewc HLP95EN.DLL,DYNALINK,"Events":"vbobjWinsockControlE"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vbobjWinsockControlS"}

The **TCP** control, invisible to the user, provides easy access to TCP network services. It can be used by Microsoft Access, Visual Basic, Visual C++, or Visual FoxPro developers. To write client or server applications you do not need to understand the details of TCP or to call low level Winsock APIs. By setting properties and invoking methods of the control, you can easily connect to a remote machine and exchange data in both directions. Events notify you of network activities.

BytesReceived Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vbproBytesReceivedPropertyC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vbproBytesReceivedPropertyX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"vbproBytesReceivedPropertyA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vbproBytesReceivedPropertyS"}

Returns the amount of data received (currently in the receive buffer). Use the **GetData** method to retrieve data.

Read-only and unavailable at design time.

Syntax

Development Tool	Syntax
Microsoft Access and Visual Basic	*object*.**BytesReceived**
Visual FoxPro	*Object*.BytesReceived
Visual C++	**long GetBytesReceived**( );

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

Return Values

Development Tool	Default Value	Data Type
Microsoft Access, Visual Basic, and Visual C++	0	Long
Visual FoxPro	0	Numeric

LocalHostName Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vbproLocalHostNamePropertyC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vbproLocalHostNamePropertyX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"vbproLocalHostNamePropertyA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vbproLocalHostNamePropertyS"}

Returns the local machine name. Read-only and unavailable at design time.

Syntax

Development Tool	Syntax
Microsoft Access and Visual Basic	*object*.**LocalHostName**
Visual FoxPro	*Object*.LocalHostName
Visual C++	**CString GetLocalHostName**( );

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

Return Values

Development Tool	Default Value	Data Type
Microsoft Access and Visual Basic	Empty	String
Visual FoxPro	Empty string	Character
Visual C++	Empty	**CString**

LocalIP Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vbproLocalIPC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vbproLocalIPX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"vbproLocalIPA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vbproLocalIPS"}

Returns the IP address of the local machine in the IP address dotted string format (xxx.xxx.xxx.xxx).Read-only and unavailable at design time.

Syntax

Development Tool	Syntax
Microsoft Access and Visual Basic	*object*.**LocalIP** = *string*
Visual FoxPro	*Object*.LocalIP
Visual C++	**CString GetLocalIP**( );

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

Data Type

Development Tool	Data Type
Microsoft Access and Visual Basic	String
Visual FoxPro	Character
Visual C++	**CString**

LocalPort Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vbproLocalPortPropertyC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vbproLocalPortPropertyX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"vbproLocalPortPropertyA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vbproLocalPortPropertyS"}

Returns or sets the local port to use. Read/Write and available at design time.

- For the client, this designates the local port to send data from. Specify port 0 if the application does not need a specific port. In this case, the control will select a random port. After a connection is established, this is the local port used for the TCP connection.
- For the server, this is the local port to listen on. If port 0 is specified, a random port is used. After invoking the **Listen** method, the property contains the actual port that has been selected.

Syntax

Development Tool	Syntax
Microsoft Access and Visual Basic	*object*.**LocalPort** = *long*
Visual FoxPro	*Object*.LocalPort[= *nPortNumber*]
Visual C++	**long GetLocalPort**( ); **void SetLocalPort**(**long** *nNewValue*);

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

Data Type

Development Tool	Data Type
Microsoft Access, Visual Basic, and	Long

Visual C++	
Visual FoxPro	Numeric

Remarks

Port 0 is often used to establish connections between computers dynamically. For example, a client that wishes to be "called back" by a server can use port 0 to procure a new (random) port number, which can then be given to the remote computer for this purpose.

RemoteHostIP Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vbproRemoteHostIPPropertyC"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vbproRemoteHostIPPropertyS"} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"vbproRemoteHostIPPropertyA"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vbproRemoteHostIPPropertyX":1}

Returns the IP address of the remote machine.

- For client applications, after a connection has been established using the **Connect** method, this property contains the IP string of the remote machine.
- For server applications, after an incoming connection request (ConnectionRequest event), this property contains the IP string of the remote machine that initiated the connection.
- For the WinSock **UDP** control, after the DataArrival event, this property contains the IP address of the machine sending the UDP data.

Syntax

Development Tool	Syntax
Microsoft Access and Visual Basic	*object*.**RemoteHostIP** = *string*
Visual FoxPro	*Object*.RemoteHostIP[= *cIPAddress*]
Visual C++	**CString GetRemoteHostIP**( );

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

Data Type

String (**CString** in Visual C++)

SocketHandle Property

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vbproSocketHandlePropertyC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vbproSocketHandlePropertyX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"vbproSocketHandlePropertyA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vbproSocketHandlePropertyS"}

Returns a value that corresponds to the socket handle the control uses to communicate with the WinSock layer. Read-only and unavailable at design time.

Syntax

Development Tool	Syntax
Microsoft Access and Visual Basic	*object*.**SocketHandle**
Visual FoxPro	*Object*.SocketHandle
Visual C++	**long GetSocketHandle**( );

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

Data Type

Development Tool	Type
Microsoft Access, Visual Basic, and Visual C++	Long
Visual FoxPro	Numeric

Remarks

This property was designed to be passed to Winsock APIs.

State Property (WinSock TCP Control)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vbproStatePropertyC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vbproStatePropertyX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"vbproStatePropertyA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vbproStatePropertyS"}

Returns the state of the control, expressed as an enumerated type. Read-only and unavailable at design time.

Syntax

Development Tool	Syntax
Microsoft Access and Visual Basic	*object*.**State**
Visual FoxPro	*Object*.State
Visual C++	**short GetState**( );

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

Data Type

Development Tool	Type
Microsoft Access, Visual Basic, and Visual C++	Integer
Visual FoxPro	Numeric

Settings

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

Constant	Value	Description
sckClosed	0	Default. Closed
sckOpen	1	Open
sckListening	2	Listening
sckConnectionPending	3	Connection pending
sckResolvingHost	4	Resolving host
sckHostResolved	5	Host resolved
sckConnecting	6	Connecting
sckConnected	7	Connected
sckClosing	8	Peer is closing the connection
sckError	9	Error

Accept Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vbmthAcceptMethodC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vbmthAcceptMethodX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"vbmthAcceptMethodA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vbmthAcceptMethodS"}

For TCP server only. This method is used to accept an incoming connection when handling a ConnectionRequest event.

Return Value

void.

Syntax

Development Tool	Syntax
Microsoft Access and Visual Basic	*object*.**Accept** *RequestID*
Visual FoxPro	*Object*.Accept(*nRequestID*)
Visual C++	**void Accept**(**long** *requestID*);

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

Arguments

Development Tool	Argument	Data Type	Description
Microsoft Access, Visual Basic, and Visual C++	*RequestID*	Long	The incoming connection request identifier. This should be the requestID passed in the ConnectionRequest event.
Visual FoxPro	*nRequestID*	Numeric	The incoming connection request identifier. This should be the requestID passed in the ConnectionRequest event.

Remarks

The **Accept** method should be used on a new control instance (other than the one that is in the listening state.)

Accept Method, ConnectionRequest Event Example

The example shows the code necessary to connect a WinSock TCP control. The code runs on the machine that is accepting the connection request. The RequestID parameter identifies the request. This is passed to the **Accept** method which accepts the particular request.

```
Private Sub WinSockTCP_ConnectionRequest(RequestID As Long)
   WinSockTCP.Accept RequestID
End Sub
```

Close Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vbmthCloseMethodC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vbmthCloseMethodX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"vbmthCloseMethodA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vbmthCloseMethodS"}

Closes a TCP connection or a listening socket for both client and server.

Return Value

void.

Syntax

Development Tool	Syntax
Microsoft Access and Visual Basic	*object.***Close**
Visual FoxPro	*Object*.Close( )
Visual C++	**void Close**( );

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

Arguments

None.

Listen Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vbmthListenMethodC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vbmthListenMethodX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"vbmthListenMethodA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vbmthListenMethodS"}

Creates a socket and sets it in listen mode.

Return Value

void.

Syntax

Development Tool	Syntax
Microsoft Access and Visual Basic	*object.***Listen**
Visual FoxPro	*Object*.Listen( )
Visual C++	**void Listen**( );

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

Arguments

None

Remarks

The ConnectionRequest event occurs when there is an incoming connection. When handling ConnectionRequest, the application should use the **Accept** method (on a new control instance) to accept the connection.

PeekData Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vbmthPeekDataMethodC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vbmthPeekDataMethodX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"vbmthPeekDataMethodA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vbmthPeekDataMethodS"}

Similar to **GetData** except **PeekData** does not remove data from the input queue.

Return Value

void.

Syntax

Development Tool	Syntax
Microsoft Access and Visual Basic	*object*.**PeekData** *Data*, [*Type*,] [*maxLen*]
Visual FoxPro	*Object*.PeekData(*cData* [, *nType*] [, *nMaxLen*])
Visual C++	**void PeekData**(**Variant*** *data*, **const Variant&** *type*, **const Variant&** *maxLen*);

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

Arguments

Development Tool	Argument	Data Type	Description
Microsoft Access, Visual Basic, and Visual C++	*Data*	Variant.	Stores retrieved data after the method returns successfully. If there is not enough data available for requested type, *data* will be set to Empty. For Output only.
	Type	Variant	Optional. Type of data to be retrieved. Default Value: vbArray + vbByte. For input only.
	maxLen	Variant	Optional. Length specifies the desired size when receiving a byte array or a string. If this argument is missing for byte array or string, all available data will be retrieved. If provided, for data types other than byte array and string, this argument is ignored. For Input only.
Microsoft Visual FoxPro	*cData*	Character.	Stores retrieved data after the method returns successfully. If there is not enough data available for requested type, *data* will be set to Empty.

		For Output only.
nType	Numeric	Optional. Type of data to be retrieved. For input only.
nMaxLen	Numeric	Optional. Length specifies the desired size. If this argument is omitted, all available data will be retrieved.

Currently, the following variant types are supported.

Type	Visual Basic
Byte	**vbByte**
Integer	**vbInteger**
Long	**vbLong**
Single	**vbSingle**
Double	**vbDouble**
Currency	**vbCurrency**
Date	**vbDate**
Boolean	**vbBoolean**
SCODE	**vbError**
String	**vbString**
Byte Array	**vbArray + vbByte**

Type	Visual C++
unsigned char	**VT_UI1**
short	**VT_I2**
long	**VT_I4**
float	**VT_R4**
double	**VT_R8**
CY	**VT_CY**
DATE	**VT_DATE**
BOOL	**VT_BOOL**
SCODE	**VT_ERROR**
BSTR	**VT_BSTR**
SAFEARRAY	**VT_ARRAY | ***

Remarks

If the type is specified as vbString, string data is converted to UNICODE before returning to the user.

SendData Method

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vbmthSendDataC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vbmthSendDataX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"vbmthSendDataA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vbmthSendDataS"}

Sends data to peer.

Return Value

void.

Syntax

Development Tool	Syntax
Microsoft Access and Visual Basic	*object*.**SendData** *data*
Visual FoxPro	*Object*.SendData(*cData*)
Visual C++	**void SendData**(**const Variant&** *data*);

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

Arguments

Development Tool	Argument	Data Type	Description
Microsoft Access, Visual Basic, and Visual C++	*data*	Variant	Data to be sent. For binary data, byte array should be used. For input only.
Visual FoxPro	*cData*	Character	Data to be sent. For input only.

Remarks

When a UNICODE string is passed in, it is converted to an ANSI string before being sent out on the network.

Close Event

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vbevtCloseEventC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vbevtCloseEventX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"vbevtCloseEventA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vbevtCloseEventS"}

Occurs when the remote computer closes the connection. Applications should use the **Close** method to correctly close the TCP connection.

Syntax

Development Tool	Syntax
Microsoft Access and Visual Basic	*object*_**Close**
Visual FoxPro	PROCEDURE *Object*.Close
Visual C++	**void** *dialogclass*::OnClose*Control*( );

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

Arguments

None.

ConnectionRequest Event

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vbevtConnectionReqestEventC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vbevtConnectionReqestEventX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"vbevtConnectionReqestEventA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vbevtConnectionReqestEventS"}

Occurs when a remote machine requests a connection.

- For WinSock TCP server only. The event is activated when there is an incoming connection request. **RemoteHostIP** and **RemotePort** properties store the information about the client after the event is activated.

Syntax

Development Tool	Syntax
Microsoft Access and Visual Basic	*object*_**ConnectionRequest** (*RequestID* **As Long**)
Visual FoxPro	PROCEDURE *Object*.ConnectionRequest LPARAMETERS *nRequestID*
Visual C++	**void** *dialogclass*::OnConnectionRequest*Control*(**long** *requestID*);

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

Arguments

Development Tool	Argument	Data Type	Description
Microsoft Access, Visual Basic, and Visual C++	*RequestID*	Long	The incoming connection request identifier. This argument should be passed to the Accept method on the second control instance. For input only.
Visual FoxPro	*nRequestID*	Numeric	The incoming connection request identifier. This argument should be passed to the Accept method on the second control instance. For input only.

Remarks

The server can decide whether or not to accept the connection. If the incoming connection is not accepted, the peer (client) will get the Close event. Use the **Accept** method (on a new control instance) to accept an incoming connection.

DataArrival Event

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vbevtDataArrivalEventC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vbevtDataArrivalEventX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"vbevtDataArrivalEventA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vbevtDataArrivalEventS"}

Occurs when new data arrives.

Syntax

Development Tool	Syntax
Microsoft Access and Visual Basic	*object*_**DataArrival** (*BytesTotal* **As Long***)*
Visual FoxPro	PROCEDURE *Object*.DataArrival LPARAMETERS *nBytesTotal*
Visual C++	**void** *dialogclass*::OnDataArrival*Control*(**long** *bytesTotal*);

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

Arguments

Development Tool	Argument	Data Type	Description
Microsoft Access, Visual Basic, and Visual C++	*BytesTotal*	Long	The total amount of data that can be retrieved. For input only.
Visual FoxPro	*nBytesTotal*	Numeric	The total amount of data that can be retrieved. For input only.

Remarks

This event will not occur if you do not retrieve all the data in one GetData call. It is activated only when there is new data. Use the **BytesReceived** property to check how much data is available at any time.

SendComplete Event

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vbevtSendCompleteEventC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vbevtSendCompleteEventX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"vbevtSendCompleteEventA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vbevtSendCompleteEventS"}

Occurs when the send buffer is empty.

Syntax

Development Tool	Syntax
Microsoft Access and Visual Basic	*object*_**SendComplete**
Visual FoxPro	PROCEDURE *Object*.SendComplete
Visual C++	**void** *dialogclass*::OnSendComplete*Control*( );

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

Arguments

None.

SendProgress Event

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vbevtSendProgressEventC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vbevtSendProgressEventX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"vbevtSendProgressEventA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vbevtSendProgressEventS"}

Notifies the user of sending progress.

Syntax

Development Tool	Syntax
Microsoft Access and Visual Basic	object_**SendProgress** (BytesSent **As Long**, BytesRemain **As Long**)
Visual FoxPro	PROCEDURE *Object*.SendProgress LPARAMETERS *nBytesSent*, *nBytesRemaining*
Visual C++	**void** *dialogclass*::OnSendProgress*Control*(**long** *bytesSent*, **long** *bytesRemaining*);

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

Arguments

Development Tool	Argument	Data Type	Description
Microsoft Access and Visual Basic	*BytesSent*	Long	The number of bytes that have been sent since the last time this event was activated. For input only.
	BytesRemain	Long	The number of bytes in the send buffer waiting to be sent. For input only.
Visual FoxPro	*nBytesSent*	Numeric	The number of bytes that have been sent since the last time this event was activated. For input only.
	nBytesRemaining	Numeric	The number of bytes in the send buffer waiting to be sent. For input only.
Visual C++	*bytesSent*	**long**	The number of bytes that have been sent since the last time this event was activated. For

		input only.
bytesRemaining	**long**	The number of bytes in the send buffer waiting to be sent. For input only.

WinSock UDP OLE Control

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vbobjWinSockUDPC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vbobjWinSockUDPX":1} {ewc HLP95EN.DLL,DYNALINK,"Properties":"vbobjWinSockUDPP"} {ewc HLP95EN.DLL,DYNALINK,"Methods":"vbobjWinSockUDPM"} {ewc HLP95EN.DLL,DYNALINK,"Events":"vbobjWinSockUDPE"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vbobjWinSockUDPS"}

The Winsock **UDP** control implements the Winsock UDP (User Datagram Protocol) for both client and server. The control represents a communication point utilizing UDP network services. It can be used to send and retrieve UDP data.

Remarks

The **UDP** control, invisible to the user, provides easy access to UDP network services. It can be used by Microsoft Access, Visual Basic, Visual FoxPro, and Visual C++ programmers. To write UDP applications you do not need to understand the details of UDP or to call low level Winsock APIs. By setting properties and calling methods on the control, you can easily connect to a remote machine and exchange data in both directions. Events are used to notify users of network activities.

GetData Method (WinSock Control)

{ewc HLP95EN.DLL,DYNALINK,"See Also":"vbmthGetDataWinSockC"} {ewc HLP95EN.DLL,DYNALINK,"Example":"vbmthGetDataWinSockX":1} {ewc HLP95EN.DLL,DYNALINK,"Applies To":"vbmthGetDataWinSockA"} {ewc HLP95EN.DLL,DYNALINK,"Specifics":"vbmthGetDataWinSockS"}

Retrieves the current block of data and stores it in a variable of type variant.

Return Value

Void

Syntax

Development Tool	Syntax
Microsoft Access and Visual Basic	*object.***GetData** *data***,** [*type,*] [*maxLen*]
Visual FoxPro	*Object.*GetData(*eData* [, *eType*] [, *eMaxLen*])

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

Arguments

Development Tool	Argument	Data Type	Description
Microsoft Access and Visual Basic	*data*	Variant	Where retrieved data will be stored after the method returns successfully. If there is not enough data available for requested type, *data* will be set to Empty.
	type	Variant	Optional. Type of data to be retrieved. Set Settings below for a list of types supported.
	maxLen	Variant	Optional. Specifies the desired size when receiving a byte array or a string. If this parameter is missing for byte array or string all available data will be retrieved. If provided, for data types other than byte array and string, this parameter is ignored.
Visual FoxPro	*eData*	Variant	Where retrieved data will be stored after the method returns successfully. If there is not enough data available for requested type, *eData* will be set to Empty.
	eType	Variant	Optional. Type of data to be retrieved. Set Settings below for a list of types supported.
	eMaxLen	Variant	Optional. Specifies the desired size when receiving a byte array or a string. If this parameter is missing for byte array or string all available

data will be retrieved. If provided, for data types other than byte array and string, this parameter is ignored.

Settings

The settings for *type* are:

Description	Visual C++	Visual Basic Type
Byte	**VT_UI1**	**vbByte**
Integer	**VT_I2**	**vbInteger**
Long	**VT_I4**	**vbLong**
Single	**VT_R4**	**vbSingle**
Double	**VT_R8**	**vbDouble**
Currency	**VT_CY**	**vbCurrency**
Date	**VT_DATE**	**vbDate**
Boolean	**VT_BOOL**	**vbBoolean**
SCODE	**VT_ERROR**	**vbError**
String	**VT_BSTR**	**vbString**
Byte Array	**VT_ARRAY|VT_UI1**	**vbArray + vbByte**

GetData Method (WinSock Control), DataArrival Event Example

The example uses the **GetData** Method in the DataArrival event of a WinSock **UDP** control. When the event occurs, the code invokes the **GetData** method to retrieve the data and store it in a string variable. The data is then written into a **TextBox** control.

```
Private Sub UDP1_DataArrival(ByVal bytesTotal As Long)
   Dim strData As String
   UDP1.GetData strData, vbString
   Text1.Text = Text1.Text & strData & vbCrLF
End Sub
```

Using the WinSock Controls

A WinSock control allows you to connect to a remote machine and exchange data between computers in both directions.

The Internet ActiveX™ controls include two WinSock controls: the WinSock **TCP** (Transmission Control Protocol) control, and the WinSock **UDP** (User Datagram Protocol) control. Both controls can be used to create client and server applications.

In simple terms, the difference between the two lies in their connection state:

- The WinSock **TCP** control is a connection-based control, and is analogous to a telephone—the user must establish a connection before proceeding.
- The WinSock **UDP** control is a connectionless control and the process by which it sends data is analogous to passing a note: a message is sent from one computer to another, but there is no explicit connection between the two.

Both the WinSock **TCP** and the WinSock **UDP** control allow data to be exchanged in both directions.

The WinSock TCP Control

Possible Uses

The WinSock controls can be used in the following scenarios:

- Create a client application that collects user information before sending it to a central server.
- Create a server application that functions as a central collection point for data from several users.

Scenario: Using the WinSock TCP Control to Send a String to a Remote Computer

The following scenario illustrates the basic mechanics of connecting two computers, in real time, over a network. One computer, the server, "listens" on a designated port. A second computer, the client, requests a connection. When the WinSock **TCP** control on the server receives the request, it creates a new instance of itself, and establishes a connection using the clone. Once a connection is established, the server and client can send data to each other. After completing a transaction, the client closes the connection, and the **TCP** control on the server destroys the instance.

On the Server computer:

1. Specify a port using the **LocalPort** property.
2. Listen using the **Listen** method.
3. Use the **Accept** Method in the ConnectionRequest event.
4. Send a string using the **SendData** method.

On the Client computer:

1. Specify a **RemoteHost** to connect to.
2. Specify a **RemotePort** to connect to.
3. Request a connection using the **Connect** method.
4. Notify user of a successful connection with the Connect event
5. Use the **GetData** method in the DataArrrival event.
6. Close the connection using the **Close** method.

Setup

To create an array of controls, place a single WinSock **TCP** control on a form and set its **Index** property to 0. Thereafter, you can load and unload instances of the control as connections are needed. In order to efficiently track the WinSock control instances, declare a single global variable that reflects the current number of instances of the control. As a connection is made, a new instance is created and the variable is incremented by one.

In Visual Basic, the code would be written in the Declarations section:

```
Public gSockInstance As Integer
```

The Visual Basic code below uses the following objects:

On the Server computer:

- WinSock **TCP** control named "sktTCPServer"
- **Form** named "frmServer"
- **CommandButton** control named "cmdSendData"
- **TextBox** control named "txtSend"

On the Client computer:

- WinSock **TCP** control named "sktTCPClient"
- **CommandButton** control named "cmdConnect"
- **CommandButton** control named "cmdCloseConnection"
- **TextBox** control named "txtReceived"
- **Label** control named " lblStatus"

Server: Specify a Port Using the LocalPort Property

The WinSock **TCP** control on the server must first be configured to listen on a particular port. Although you can designate any number, some numbers are reserved for certain protocols. For example, HTML browsers use port number 80. The code below uses the port number 1007 since it is not reserved for any other protocol use.

```
sktTCPsvr.LocalPort = 1007
```

Server: Listen Using the Listen Method

Besides specifying a port to listen on, the TCP control must also be "listening" for the client computer. The following code shows how to do this using the **Listen** method, through the form's Load event:

```
Private Sub frmServer_Load()
   sktTCPServer.LocalPort = 1007 ' Set the local port.
   sktTCPServer.Listen ' Use the Listen method.
End Sub
```

Client: Request a Connection Using the Connect Method

To begin a transaction, a connection must be made first. To accomplish this, the client machine uses the **Connect** method which takes two arguments, **RemoteHost** and **RemotePort** properties. The **RemoteHost** property specifies a machine to which the user wants to connect. This property can be either a string, the "friendly name" for the server computer, or an Internet Protocol (IP) address, a unique string of numbers that specifies the remote computer. The code below sets the **RemoteHost** and **RemotePort**, then invokes the **Connect** method.

```
Private Sub cmdConnect_Click()
   With sktTCPClient
      .RemoteHost = "123.123.101.201"
      .RemotePort = 1007
```

```
      .Connect
   End With
End Sub
```

Alternatively, you can use the **Connect** method alone and supply the **RemoteHost** and **RemotePort** as optional arguments.

```
Private Sub cmdConnnect_Click()
   sktTCPClient.Connect "123.123. 101.201", 1007
End Sub
```

Server: Use the Accept Method in the ConnectionRequest Event

When the server computer receives a connection request from the client computer, the ConnectionRequest event occurs. Use this event to respond with the **Accept** method which accepts the server application's connection request.

The ConnectionRequest event passes a single argument, the requestID, that uniquely identifies the request. By no coincidence, the **Accept** method requires one argument, which should be the value of the requestID.

The code below executes when the server computer receives a connection request. In the ConnectionRequest event, the code first increments the global variable **gSockInstance** and uses the new value to load a new instance of the control. Then the code passes the value of the requestID to the **Accept** method which establishes the connection.

```
Private Sub sktServer_ConnectionRequest _
(Index As Integer, ByVal requestID As Long)
   ' Increment the global variable.
   gSockInstance = gSockInstance + 1
   Load sktTCPServer(gSockInstance)
   sktTCPServer(gSockInstance).Accept requestID
End Sub
```

Client: Notify User of a Successful Connection with the Connect Event

Upon a successful connection, the **Connect** event is triggered on the client computer. The code below uses this event with the **State** property to notify the user of the successful connection.

```
Private Sub sktClient_Connect()
   If sktTCPClient.State = sckConnected Then
   ' Presuming a Statusbar exists, with one panel.
   lblStatus.Caption ="Connection Successful!"
   End If
End Sub
```

Server: Send a String Using the SendData Method

Once a connection has been established, you can send data to the remote computer with the **SendData** method. The code below sends a string from the server to the client.

```
Private Sub cmdSendData_Click()
   sktTCPClient.SendData "This is how we begin."
End Sub
```

Client: Use the GetData Method in the DataArrrival Event

On the client computer, the sent message triggers the DataArrival event. When this event occurs, use the **GetData** method to retrieve the data, as shown below.

```
Private Sub sktTCPServer_DataArrival _
(Index As Integer, ByVal bytesTotal As Long)
   Dim vtData ' Declare a variant to hold the data.
```

```
   sktTCPServer(Index).GetData vtData, vbString
   txtReceived.Text = vtData ' Display the data.
End Sub
```

Client: Close the Connection Using the Close Method.

After completing a transaction, the client uses the **Close** method to close the connection.

```
Private Sub cmdCloseConnection_Click()
   sktTCPClient.Close
End Sub
```

This triggers the Close event on the server, which can then unload the instance of the control, and decrement the global variable.

```
Private Sub sktTCPServer_Close (Index As Integer)
   sktTCPServer(Index).Close
   Unload sktTCPServer(Index) ' Unload the instance.
   gSockInstance = gSockInstance - 1 ' Decrement the
                                     ' variable.
End Sub
```

WinSock UDP Control

Scenario: Using the WinSock UDP Control to Broadcast a Message

The WinSock **UDP** control behaves much like the WinSock **TCP** Control. However, the User Datagram Protocol is a *connectionless* protocol. Unlike the TCP control, which must have an established connection before it can send or receive data, the **UDP** control doesn't have either a **Connect** or a **Listen** method. Instead, the control needs only to know which RemotePort and RemoteHost to send data to. In this, it behaves somewhat like a radio—the control sends a message to the other computer, but it doesn't know if the other computer has received the message.

To send a message from one computer to another:

On the sending computer:

1. Set the **RemoteHost** and **RemotePort** properties.
2. Send a message using the **SendData** method.

On the receiving computer:

1. Set the **LocalPort** property
2. Use the **GetData** method in the DataArrival event to retrieve the message.

Setup

The Visual Basic code below uses the following objects:

On the sending computer:

- **Form** named "frmSend"
- **UDP** control named "udpSender"
- **CommandButton** control named "cmdSendData"
- **TextBox** control named "txtSend"

On the receiving computer:

- **Form** named "frmReceiver"
- **UDP** control named "udpReceiver"
- **TextBox** control named "txtReceived"

Sender: Set the RemoteHost and RemotePort Properties

On the computer which will send the data, it's only necessary to specify the name of the receiving computer and its port. This is done with the **RemoteHost** and **RemotePort** properties, as shown below:

```
Private Sub frmSend_Load ()
   udpSender.RemoteHost = "123.123.101.201"
   udpSender.RemotePort = 1007
End Sub
```

Receiver: Set the LocalPort Property

On the computer which will receive the message, it's only necessary to specify a **LocalPort**—this should correspond to the **RemotePort** property on the sending computer.

```
Private Sub frmReceiver_Load()
   udpReceiver.LocalPort = 1007
End Sub
```

Sender: Send a Message Using the SendData Method

On the computer sending the data, use the **SendData** method.

```
Private Sub cmdSendData_Click()
   udpSender.SendData "Calling all cars..."
End Sub
```

Receiver: Use the GetData Method in the DataArrival Event to Retrieve the Message

On the receiving computer, use the DataArrival event to process the message. In this case, the message is put in a **TextBox** control

```
Private Sub udpReceiver_DataArrival _
(ByVal bytesTotal As Long)
   Dim vtData ' Declare a variant to hold the data.
   udpReceiver.GetData vtData, vbString
   txtReceived.Text = vtData ' Display the message.
End Sub
```