Help for Socket/X

Properties Events Methods Interfaces

How To Buy This Software

Order Form

Getting Custom Controls Written

Licensing Information

Description

Socket/X provides you with full access to the power of Windows Sockets, making it easy to write TCP/IP or UDP client and server software. Socket/X comes in ActiveX control and COM object forms. Because Socket/X is both an ActiveX control and a COM object, you can use it nearly anywhere -- ASP pages, Visual Basic applications, Visual C++ applications, anywhere that supports either COM objects or ActiveX controls.

Socket/X provides complete support for Windows and Visual Basic's event-driven programming model and lets you do nearly anything that can be done with WinSock. Socket/X goes beyond the normal Visual Basic event-driven model by providing a scheme for Fast Notifications. Fast Notifications allow your program to quickly receive events through simple functions, rather than going through the lengthy process to fire an event. Fast Notifications considerably reduce the amount of time required to handle an event and only require a couple extra lines of code on your part. Want to stick with events? No problem! Those are still supported.

You can also use Socket/X in a "blocking" mode. This enables you to use Socket/X where events are callbacks are impractical (such as ASP pages).

Socket/X supports stream-based and datagram sockets. Stream-based sockets provide sequenced, reliable, full-duplex, connection-oriented byte streams. With stream-based sockets your data is guaranteed to arrive. This uses the Transmission Control Protocol (TCP) for the Internet address family.

Datagram sockets are connectionless, unreliable packets (typically small) of a fixed maximum length. These kinds of sockets are good for broadcasting large quantities of small pieces of information to a lot of clients. This style of socket uses the User Datagram Protocol (UDP) for the Internet address family.

Adding The ActiveX Control To Your Project

The ActiveX control (SocketX.OCX) is added to a project just like any other 32-bit control. If using VB5 or VB6, select "Components..." from the "Project" menu and select "Mabry SocketX Control" from the list of available controls.

Adding The COM Object To Your Project

To instantiate Socket/X as a COM Object (SocketX.DLL), choose "References..." from the "Project" menu and select "Mabry SocketX COM Object" when using VB5 or VB6. **Tip:** be sure to select the COM Object and not the control.

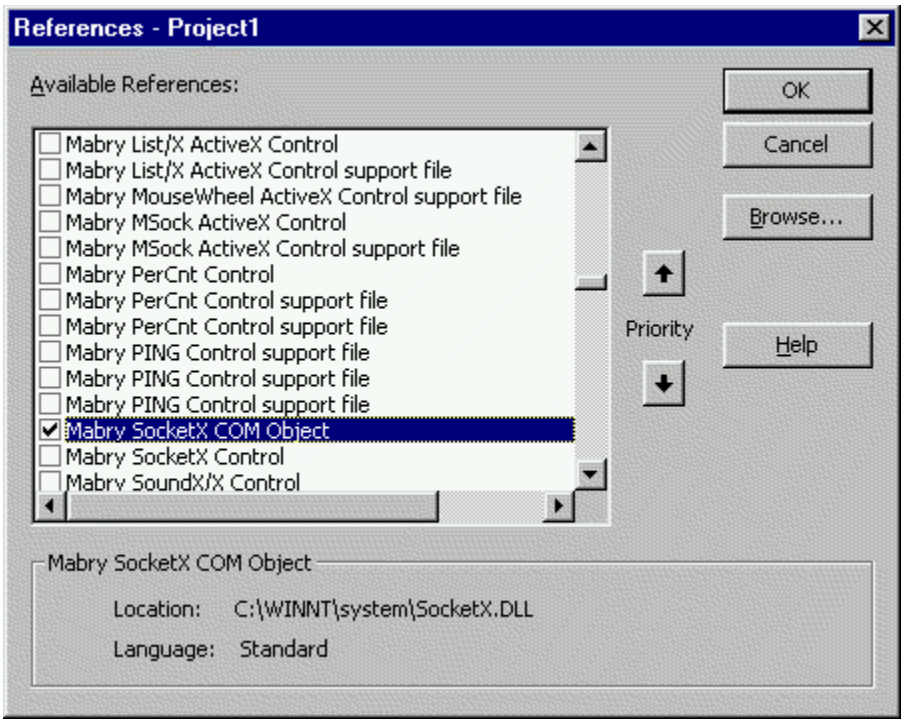

Figure 1

Once SocketX is referenced in the project, you declare and set the object as shown below. After the object variable has been set, you can use it just as you would if you placed a control on a form.

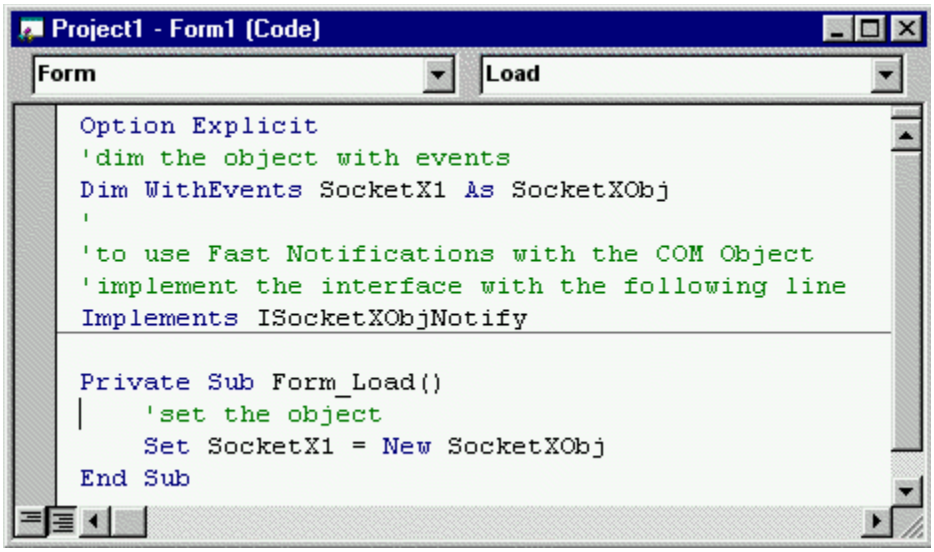

Figure 2

Using Fast Notification

Fast Notifications considerably reduces the amount of time required to handle events. When you implement the SocketX Fast Notifications interface directly in your application, the SocketX ActiveX control or COM object directly notifies your application rather than going through the overhead of the Visual Basic event model.

To implement Fast Notifications for the COM Object, use the Implements statement as shown in Figure 2. This will allow your application to directly access the events of the ISocketXObjNotify class.

To implement Fast Notifications for the ActiveX control, use Implements ISocketXCtlNotify. This will

allow your application to directly access the events of the ISocketXCtlNotify class.

Visual Basic 4 (32-bit) Notes

When inserting the ActiveX control in a VB4-32 project, select "Custom Controls..." from the "Tools" menu and select "Mabry SocketX Control".

If using the COM Object in a VB4-32 Project, "References..." is found under the "Tools" menu.

Note: VB4 does not support Implements, so the Fast Notifications interface cannot be implemented in VB4-32 applications.

ActiveX Compatibility

VB 4.0 (32-bit), 5.0 and 6.0

ActiveX Built With

Microsoft Visual C++ v6

ActiveX - Required DLLs

None. This is a light ActiveX control that requires no extra DLLs to operate. This means that your installation packages will be smaller. And, your web pages will load faster, since there are no extra DLLs to download.

Distribution Note When you develop and distribute an application that uses this control, you should install the control file into the user's Windows SYSTEM directory. The control file has version information built into it. So, during installation, you should ensure that you are not overwriting a newer version.

Close

Socket/X Properties

Properties that have special meaning for this control or that only apply to this control are marked with an asterisk (*).

- *__AcceptTimeout__ Property
- *__Blocking__ Property
- *__BlockingMode__ Property
- *__BroadcastEnabled__ Property
- *__BytesReceived__ Property
- *__BytesSent__ Property
- *__EventMask__ Property
- *__KeepAliveEnabled__ Property
- *__LastError__ Property
- *__LastErrorString__ Property
- *__LastMethod__ Property
- *__LibraryName__ Property
- *__LingerEnabled__ Property
- *__LingerMode__ Property
- *__LingerTime__ Property
- *__Listening__ Property
- *__LocalAddress__ Property
- *__LocalPort__ Property
- *__NotificationObject__ Property
- *__OutOfBandEnabled__ Property
- *__ReceiveBuffer__ Property
- *__ReceiveBufferSize__ Property
- *__ReceiveTimeout__ Property
- *__RemoteAddress__ Property
- *__RemoteName__ Property
- *__RemoteNameAddrXlate__ Property
- *__RemotePort__ Property
- *__ReuseAddressEnabled__ Property
- *__RouteEnabled__ Property
- *__SendBuffer__ Property
- *__SendBufferSize__ Property
- *__SendTimeout__ Property
- *__Socket__ Property
- *__SocketAddress__ Property
- *__SocketPort__ Property
- *__SocketType__ Property
- *__State__ Property
- *__StateString__ Property
- *__TCPNoDelayEnabled__ Property
- *__Version__ Property

*__WSAVersion__ Property
*__WSDescription__ Property
*__WSMaxSockets__ Property
*__WSMaxUDPSize__ Property

Close

Socket/X Events

Events that have special meaning for this control or that only apply to this control are marked with an asterisk (*).

*__Accept__ Event

*__Close__ Event

*__Connect__ Event

*__Done__ Event

*__OutOfBandData__ Event

*__Receive__ Event

*__Send__ Event

Close

Socket/X Methods

Methods that have special meaning for this control or that only apply to this control are marked with an asterisk (*).

AboutBox Method
*__**Bind** Method__
*__**Close** Method__
*__**Connect** Method__
*__**Create** Method__
*__**Listen** Method__
*__**Receive** Method__
*__**ReceiveFrom** Method__
*__**Send** Method__
*__**SendTo** Method__

Close

Socket/X Interfaces

Interfaces that have special meaning for this control or that only apply to this control are marked with an asterisk (*).

*__SocketXCtlNotify__ Interface

Close

Socket/X SocketXCtlNotify Interface Events

Events that have special meaning for this control or that only apply to this control are marked with an asterisk (*).

*__Accept__ Event

*__Close__ Event

*__Connect__ Event

*__Done__ Event

*__OutOfBandData__ Event

*__Receive__ Event

*__Send__ Event

How To Buy This Software

CREDITS

Socket/X was written by Zane Thomas.

CONTACT INFORMATION

Orders, inquiries, technical support, questions, comments, etc. can be sent to mabry@mabry.com on the Internet. Our mailing address/contact information is:

Mabry Software, Inc.
503 316th Street Northwest
Stanwood, WA 98292

Sales: 1-800-99-MABRY (U.S. Only)
Voice: 360-629-9278
Fax: 360-629-9278
Web: http://www.mabry.com

COST

The price of Socket/X (control only) is US$50 (US$55 for International orders). The cost of Socket/X and the C/C++ source code (of the control itself) is US$149 (US$154 for International orders).

Prices are subject to change without notice.

Printed manuals are available at US$12.50 per copy.

DELIVERY METHODS

We can ship this software to you via air mail and/or e-mail.

Air Mail - you will receive diskettes, a printed manual (if purchased), and printed receipt if you choose this delivery method. The costs are:

US$10.00	US Priority Mail
US$15.00	Airborne Express 2nd Day (US deliveries only)
US$20.00	Airborne Express Overnight (US deliveries only)
US$20.00	Global Priority Mail (Int'l deliveries only; Western Europe, Pacific Rim and Canada only)
US$45.00	International Airborne Express (Int'l deliveries only)

E-Mail - We can ship this package to you via e-mail. You need to have an e-mail account that can accept large file attachments (which includes CompuServe, AOL, and most Internet providers). We will e-mail a receipt to you.

Be sure to include your full mailing address with your order. Sometimes (on the Internet) the package cannot be e-mailed, so we are forced to send it through the normal mails.

CompuServe E-Mail - CompuServe members can use the software registration forum (GO SWREG) to register this package. Socket/X's SWREG ID number is 15751. The source code version's ID number is 15752. PLEASE NOTE: When you order through SWREG, we send the registered package to your CompuServe account (not your Internet or AOL account) within a few hours.

ORDER / PAYMENT METHODS

You can order this software by phone, fax, e-mail, mail. For your convenience, an order form has been provided that you can print out directly from this help file.

Please note that orders must include all information that is requested on our order form. Your shipment WILL BE DELAYED if we have to contact you for additional information (such as phone number, street address, etc.).

You can pay by credit card (VISA, MasterCard, American Express, Discover, NOVUS), check (U.S. dollars drawn on a U.S. bank), cash, International Money Order, International Postal Order, Purchase Order (established business entities only - terms net 30), or wire transfer.

WIRE TRANSFER INFORMATION

Here is the information you need regarding our account for a wire funds transfer:

Bank Name:	SeaFirst - Stone Way Branch
Bank Address:	3601 Stone Way North Seattle, WA 98103
Bank Phone:	206-585-4951
Account Name:	Mabry Software, Inc.
Routing Number:	12000024
Account Number:	16311706

If you are paying with a wire transfer of funds, please add US$25.00 to your order. This is the fee that SeaFirst Bank charges Mabry Software. Also, please ADD ANY ADDITIONAL FEES THAT YOUR BANK MAY CHARGE for wire transfer service. If you are paying with a wire transfer, we must have full payment deposited to our account before we can ship your order.

Socket/X Order Form

Use the Print Topic... command from the File menu to print this order form.

Mail this form to: Mabry Software, Inc.
503 316th Street Northwest
Stanwood, WA 98292

Phone: 360-629-9278
Fax: 360-629-9278
Internet: mabry@mabry.com
Web: www.mabry.com

Where did you get this copy of Socket/X?

_____

Name: _____

Ship to: _____

_____

_____

_____

Phone: _____

Fax: _____

E-Mail: _____

Credit Card #: _____ exp. _____

P.O. # (if any): _____ Signature _____

qty ordered ____ REGISTRATION
$50.00 ($55.00 international). Check or money order in U.S. currency drawn on a U.S. bank. Add $10.00 per order for shipping and handling. Add $12.50 per printed manual.

qty ordered ____ SOURCE CODE AND REGISTRATION
$149.00 ($154.00 international). Check or money order in U.S. currency drawn on a U.S. bank. Add $10.00 per order for shipping and handling. Add $12.50 per printed manual.

Close Copy Print

Error Codes

Constant	Value	Description
	0	No error.
WSAEINTR	10004	System level interrupt interrupted socket operation.
WSAEBADF	10009	Generic error for invalid format, bad format.
WSAEACCES	10013	Generic error for access violation.
WSAEFAULT	10014	Generic error for fault.
WSAEINVAL	10022	Generic error for invalid format, entry, etc.
WSAEMFILE	10024	Generic error for file error.
	10025	The IP address provided is not valid or the host specified by the IP does not exist.
WSAENOTSOCK	10038	Invalid socket or not connected to remote.
WSAEADDRINUSE	10048	The specified address is already in use.
WSAEADDRNOTAVAIL	10049	The specified address is not available.
WSAENETDOWN	10050	The connected network is not available.
WSAENETUNREACH	10051	The connected network is not reachable.
WSAENETRESET	10052	The connected network connection has been reset.
WSAECONNABORTED	10053	The current connection has been aborted by the network or intermediate services.
WSAECONNRESET	10054	The current socket connection has been reset.
WSAENOTCONN	10057	The current socket has not been connected.
WSAESHUTDOWN	10058	The connection has been shutdown.
WSAETIMEDOUT	10060	The current connection has timed out.
WSAECONNREFUSED	10061	The requested connection has been refused by the remote host.
WSAENAMETOOLONG	10063	Specified host name is too long.
WSAEHOSTDOWN	10064	Remote host is currently unavailable.
WSAEHOSTUNREACH	10065	Remote host is currently unreachable.
WSASYSNOTREADY	10091	Remote system is not ready.
WSAVERNOTSUPPORTED	10092	Current socket version not supported by application.
WSANOTINITIALISED	10093	Socket API is not initialized.
WSAEDISCON	10101	Socket has been disconnected.

See Also

Listen Method

Accept Event

See Also

Connect Method
Listen Method
LocalAddress Property
LocalPort Property

See Also

BlockingMode Property

See Also

Blocking Property

See Also

Create Method

See Also

Send Event

Send Method

See Also

LingerEnabled Property
LingerMode Property
LingerTime Property

See Also

Close Method

See Also

Connect Method

See Also

Close Method
RemoteAddress Property
RemotePort Property

See Also

EventMask Property
LocalAddress Property
LocalPort Property
SocketType Property

See Also

- **Create** Method
- **Accept** Event
- **Close** Event
- **Connect** Event
- **OutOfBandData** Event
- **Receive** Event
- **Send** Event

See Also

Connect Method

Create Method

See Also

LastErrorString Property

See Also

LastError Property

See Also

Close Method
LingerMode Property
LingerTime Property

See Also

Close Method
LingerEnabled Property
LingerTime Property

See Also

Close Method
LingerEnabled Property
LingerMode Property

See Also

Create Method
LocalAddress Property
LocalPort Property

See Also

Listen Method

Accept Event

See Also

Connect Method

LocalPort Property

See Also

Connect Method

LocalAddress Property

See Also

OutOfBandEnabled Property

ReceiveFrom Method

See Also

OutOfBandData Event

See Also

Receive Method

ReceiveFrom Method

See Also

Receive Event

ReceiveBuffer Property

See Also

Receive Event

Receive Method

ReceiveFrom Method

See Also

ReceiveBuffer Property

SendBufferSize Property

See Also

RemoteAddress Property

RemotePort Property

See Also

Receive Event
Receive Method
ReceiveBuffer Property
ReceiveFrom Method

See Also

RemoteName Property

RemoteNameAddrXlate Property

See Also

RemoteAddress Property

RemoteNameAddrXlate Property

See Also

RemoteAddress Property

RemoteName Property

See Also

LocalPort Property

RemoteAddress Property

See Also

LocalAddress Property

LocalPort Property

See Also

Connect Method

See Also

BytesSent Property
Send Event
SendBuffer Property

See Also

Send Method

SendBuffer Property

SendTo Method

See Also

Send Event
ReceiveBuffer Property
Send Method
SendTo Method

See Also

ReceiveBufferSize Property

SendBuffer Property

See Also

Send Event

Send Method

SendBuffer Property

SendTo Method

See Also

RemoteAddress Property
RemotePort Property
Send Method
SendBuffer Property

See Also

Listen Method

Accept Event

See Also

Bind Method

Connect Method

SocketPort Property

See Also

Bind Method

Connect Method

SocketAddress Property

See Also

Create Method

See Also

StateString Property

See Also

State Property

See Also

Connect Method

See Also

LastError Property

LastErrorString Property

LastMethod Property

AcceptTimeout Property

Description

Time to wait for an Accept to complete.

Syntax

object.**AcceptTimeout**

The syntax of the **AcceptTimeout** property has these parts:

Part	Description
object	A Socket/X control.

Remarks

An Accept can take some time to complete. If the AcceptTimeout value is non-zero and if AcceptTimeout seconds elapse without the Accept method completing, an error will be returned.

Data Type

Integer

Bind Method

See Also Error Codes

Description

Associates a local address with the socket.

Syntax

object.**Bind***LocalAddress*,*LocalPort*

The syntax of the **Bind** method has these parts:

Part	Description
object	Required. A Socket/X control.
LocalAddress	Optional. A string expression that specifies the address of the server to listen to. If not specified, the LocalAddress property is used.
LocalPort	Optional. An integer that specifies the port number, on the server, to listen to. If not specified, the LocalPort property is used.

Remarks

This method is used on an unconnected datagram or stream socket, before subsequent Connect or Listen calls. Before it can accept connection requests, a listening server socket must select a port number and make it known to Windows Sockets by calling Bind. Bind establishes the local association (host address/port number) of the socket by assigning a local name to an unnamed socket.

BlockingMode Property

See Also

Description

Controls message processing during Blocking calls.

Syntax

object.**BlockingMode**

The syntax of the **BlockingMode** property has these parts:

Part	Description
object	A Socket/X control.

Remarks

There are two different block modes to choose from, soxTrueBlocking and soxPseudoBlocking. If you have Blocking enabled and TrueBlocking mode selected, then blocking calls will process only WM_PAINT messages for your program. All other messages will be ignored.

If you choose PseudoBlocking mode then all of your program's messages will be processed, including mouse, keyboard, and so on. PseudoBlocking allows full UI interactivity while blocking operations are in progress, but also requires that you exercise considerable care to ensure that you do not call any other blocking functions while one is in progress. If you fail to handle this correctly, errors will result (which you may choose to handle with On Error Goto or similar statements).

PseudoBlocking is handy when you want to write socket handling code that effectively functions as a second thread within your program. For instance, you can have the following code executing:

```
SocketX1.Receive
InputBuf = SocketX1.ReceiveBuffer
SocketX1.SendBuffer = "output data"
SocketX1.Send
SocketX1.Receive
```

This code can execute while you continue to service UI requests via menu items, commmand buttons,and so on.

Note that PseudoBlocking mode does *not* provide unlimited multi-threading capabilities. Consider what would happen if you had two bits of socket code running at the same time. We'll call them ThreadA and ThreadB. If ThreadA makes a blocking call and then something in your program (like user input or a timer) triggers ThreadB and it too blocks on a socket call, then ThreadA will *not* return from its blocking call until ThreadB finishes *all* of its socket code and returns out of the event procedure that initiated it. This is a direct consequence of the way Windows works and there is nothing that can be done in SocketX to change it. The only way to handle multiple sockets is to use non-blocking mode.

Data Type

Integer

Blocking Property

See Also

Description

Enables/disables blocking mode.

Syntax

object.**Blocking**

The syntax of the **Blocking** property has these parts:

Part	Description
object	A Socket/X control.

Remarks

When Blocking is set to False, methods execute asynchronously. When Blocking is set to True, methods do not return until finished (synchronous). The BlockingMode property controls the type of events that will be processed while waiting for blocking calls to return.

Data Type

Boolean

BroadcastEnabled Property

See Also Error Codes

Description

Enables the transmission of broadcast packets.

Syntax

object.**BroadcastEnabled** [= *boolean*]

The syntax of the **BroadcastEnabled** property has these parts:

Part	Description
object	A Socket/X control.
boolean	A boolean expression that indicates whether packets may be broadcast.

Remarks

Broadcast packets can only be sent over datagram sockets.

Data Type

Boolean

BytesReceived Property

Description

Number of bytes received by last Read or ReadFrom method call.

Syntax

object.**BytesReceived**

The syntax of the **BytesReceived** property has these parts:

Part	Description
object	A Socket/X control.

Remarks

This property returns the number of bytes received by last Read or ReadFrom method. This property is read-only and only available at run-time.

Data Type

Long

BytesSent Property

See Also Error Codes

Description

Holds the number of bytes sent with the Send method.

Syntax

object.**BytesSent**

The syntax of the **BytesSent** property has these parts:

Part	Description
object	A Socket/X control.

Remarks

After the Send method is called, the BytesSent property will contain the number of bytes queued for transmission by the socket protocol stack. If the protocol stack queue is either full when the Send method is called, or becomes full as a result of some of the Send data being queued, then BytesSent can be examined to determine how many bytes remain in the SendBuffer to be sent.

Microsoft's WinSock stack does not queue partial messages and so, if the Send method fails, the WSAEWOULDBLOCK error (10035) is thrown and BytesSent will be zero. However, in the future Microsoft may decide to queue some of the bytes in the SendBuffer property. And, of course, protocol stacks from other vendors may do so as well. If only some of the bytes are sent, then an error is thrown with the value of the Err variable set to 20000. BytesSent will be some value greater than zero and less than the number of bytes in SendBuffer. In this case, you will need to wait for the Send event and then send the remainder of the bytes. Be sure to do something like:

```
Ctl.SendBuffer = right(len(Ctl.SendBuffer) - Ctl.BytesSent)
Ctl.Send
```

Data Type

Long

Close Method

See Also Error Codes

Description

Closes a created socket.

Syntax

object.**Close**

The syntax of the **Close** method has these parts:

Part	Description
object	Required. A Socket/X control.

Remarks

The semantics of Close are affected by the LingerEnabled, LingerMode, and LingerTime properties.

If the LingerEnabled property is False, then Close will return immediately. However, any data queued for transmission will be sent, if possible, before the underlying socket is closed. This is also called a graceful disconnect.

If LingerEnabled is True and if LingerMode is non-zero, the socket will close gracefully. Control will return from the call after the socket finishes closing, if LingerTime is non-zero.

If LingerMode is zero or LingerTime is zero, the socket will be closed immediately.

Connect Method

See Also Error Codes

Description

Establishes a connection to a peer.

Syntax

object.**Connect***RemoteAddress*,*RemotePort*

The syntax of the **Connect** method has these parts:

Part	Description
object	Required. A Socket/X control.
RemoteAddress	Optional. A string expression that specifies the address of the server to connect to. If not specified, the RemoteAddress property is used.
RemotePort	Optional. An integer that specifies the port number, on the server, to connect to. If not specified, the RemotePort property is used.

Remarks

The Connect method is used to establish a connection to a server. The *RemotePort* and *RemoteAddress* parameters (or properties, when the optional parameters are not passed) specify the server to connect to.

Connect is an asynchronous operation and the Connect method will most likely return a WSAEWOULDBLOCK (10035) error. When connecting to local server ports, this error may not occur. If the WSAEWOULDBLOCK error is returned, you should wait until the Connect event is fired before attempting to communicate with the server socket.

Create Method

See Also Error Codes

Description

Creates and opens a socket.

Syntax

object.**Create***LocalPort*,*SocketType*,*EventMask*,*LocalAddress*

The syntax of the **Create** method has these parts:

Part	Description
object	Required. A Socket/X control.
LocalPort	Optional. An integer that specifies the port number to create.
SocketType	Optional. An integer that specifies the type of socket: stream or datagram.
EventMask	Optional. An integer that specifies the events to fire for this socket.
LocalAddress	Optional. A string expression that specifies the address of the socket to create.

Remarks

The Create method creates and opens a socket using the specified port and IP address.

SocketType can be:

Constant	Value	Description
soxStream	0	Stream socket
soxDatagram	1	Datagram socket

EventMask can be any combination of the following:

Constant	Value	Description
soxReadEvent	1	Receive event fired when data is received.
soxWriteEvent	2	Send event fired when the socket is ready to send data.
soxOOBEvent	4	OutOfBandData event fired when out-of-band data is received.
soxAcceptEvent	8	Accept event fired when a client connection has been accepted.
soxConnectEvent	16	Connect event fired when a socket connects to a server.
soxCloseEvent	32	Close event fired when the socket closes.
soxAllEvents	63	Enables all of the socket events.

Done Event

Description

Fired when a method finishes.

Syntax

Sub *object*_**Done(**[*index* **As Integer**]**)**

The syntax of the **Done** event has these parts:

Part	Description
object	A Socket/X control.
index	An integer that identifies a control if it's in a control array.

Remarks

The Done event is fired regardless of whether Blocking is True or False.

SocketXCtlNotify Interface Done Event

See Also

Description

Called when the control completes a method.

Syntax

Sub *object*_**Done(**[*index* **As Integer**]**)**

The syntax of the **Done** event has these parts:

Part	Description
object	Required. An instance of a SocketXCtlNotify interface.

Remarks

This method is called when the data request completes. If the ErrorCode parameter is zero, the operation completed correctly.

LastError Property

See Also Error Codes

Description

Returns last error code.

Syntax

object.**LastError**

The syntax of the **LastError** property has these parts:

Part	Description
object	A Socket/X control.

Remarks

This property holds the most recent error number returned by this control. Any time any method is executed, this property is reset to zero (0) prior to executing the method.

Data Type

Integer

EventMask Property

See Also Error Codes

Description

Enables the socket control's notification events.

Syntax

object.**EventMask** [= *integer*]

The syntax of the **EventMask** property has these parts:

Part	Description
object	A Socket/X control.
integer	An integer that specifies the events fired for the current socket.

Remarks

One or more of the following constants can be used to enable socket events:

Constant	Value	Description
soxReadEvent	1	Receive event fired when data is received.
soxWriteEvent	2	Send event fired when the socket is ready to send data.
soxOOBEvent	4	OutOfBandData event fired when out-of-band data is received.
soxAcceptEvent	8	Accept event fired when a client connection has been accepted.
soxConnectEvent	16	Connect event fired when a socket connects to a server.
soxCloseEvent	32	Close event fired when the socket closes.
soxAllEvents	63	Enables all of the socket events.

Data Type

Integer

KeepAliveEnabled Property

See Also Error Codes

Description

Enables the transmission of keep-alive packets.

Syntax

object.**KeepAliveEnabled** [= *boolean*]

The syntax of the **KeepAliveEnabled** property has these parts:

Part	Description
object	A Socket/X control.
boolean	A boolean expression that indicates whether keep-alive packets are sent to the server.

Remarks

Some servers will drop a connection for inactivity. The KeepAliveEnabled property can be set to True to cause keep-alive packets to be sent to the server.

Data Type

Boolean

LastErrorString Property

See Also

Description

Holds a description of the last error number reported.

Syntax

object.**LastErrorString**

The syntax of the **LastErrorString** property has these parts:

Part	Description
object	A Socket/X control.

Remarks

This property contains a user-friendly version of the result of the last method executed. It is zero if the last method completed without error. Otherwise, it contains an error code.

This property is read-only and only available at run-time.

Data Type

String

LastMethod Property

Description

Returns the last method executed.

Syntax

object.**LastMethod**

The syntax of the **LastMethod** property has these parts:

Part	Description
object	A Socket/X control.

Remarks

This property returns an integer which represents the last method executed.

This property is read-only and only available at run-time.

Data Type

Integer

LibraryName Property

Description

Specifies the name of the socket stack DLL.

Syntax

object.**LibraryName**

The syntax of the **LibraryName** property has these parts:

Part	Description
object	A Socket/X control.

Remarks

The default value for this property is "wsock32.dll", which is the socket stack that Microsoft ships with 32-bit systems. If you have a different socket stack provider, you can use the provider's socket stack by specifying the provider's DLL name.

Data Type

String

LingerEnabled Property

See Also Error Codes

Description

Affects the way a socket is closed.

Syntax

object.**LingerEnabled** [= *boolean*]

The syntax of the **LingerEnabled** property has these parts:

Part	Description
object	A Socket/X control.
boolean	A boolean expression that determines whether or not the socket lingers prior to closing.

Remarks

Please see the Close method for a description of how this property affects socket closing.

Data Type

Boolean

LingerMode Property

See Also Error Codes

Description

LingerMode affects the way the Close method works.

Syntax

object.**LingerMode** [= *integer*]

The syntax of the **LingerMode** property has these parts:

Part	Description
object	A Socket/X control.
integer	An integer that determines if lingering is on or off.

Remarks

Please see the Close method for details on the use of this property. This property may be set to the following values:

Constant	Value	Description
soxLingerModeOff	0	Linger mode is disabled.
soxLingerModeOn	1	Linger mode is enabled.

Data Type

Integer

LingerTime Property

See Also Error Codes

Description

The LingerTime property affects the way the Close method works.

Syntax

object.**LingerTime** [= *time*]

The syntax of the **LingerTime** property has these parts:

Part	Description
object	A Socket/X control.
time	An integer that determines how long the socket lingers while closing.

Remarks

Please see the Close method for details on the use of this property.

Data Type

Integer

Listen Method

See Also Error Codes

Description

This method enables a socket to listen for incoming connection requests.

Syntax

object.**Listen**

The syntax of the **Listen** method has these parts:

Part	Description
object	Required. A Socket/X control.

Remarks

Listen is the method to use if you want to accept client connection requests. In other words, this is how you create a server. Listen applies only to sockets that support connections, that is, those of type soxStream (see the Create method). This socket is put into passive mode where incoming connections are acknowledged and queued, pending acceptance by the process.

Listening Property

See Also Error Codes

Description

Returns True when a socket is waiting (listening) for a client to connect.

Syntax

object.**Listening**

The syntax of the **Listening** property has these parts:

Part	Description
object	A Socket/X control.

Remarks

When a socket is waiting for a client to connect, it is listening, and the Listening property will be true.

This property is read-only and only available at run-time.

Data Type

Boolean

LocalAddress Property

See Also Error Codes

Description

Specifies the IP address of the socket.

Syntax

object.**LocalAddress** [= *string*]

The syntax of the **LocalAddress** property has these parts:

Part	Description
object	A Socket/X control.
string	A string expression that specifies an IP address.

Remarks

LocalAddress and LocalPort are used when invoking the Create method to explicitly create a socket on the local machine. LocalAddress is used to specify the IP address of the local machine. LocalPort should always be set to zero when creating a client application so the stack picks a free port. When creating a server (listening socket), the LocalPort property would be set to the port on which the server should listen for incoming connections.

Data Type

String

LocalPort Property

See Also Error Codes

Description

Specifies the socket's port number.

Syntax

object.**LocalPort** [= *integer*]

The syntax of the **LocalPort** property has these parts:

Part	Description
object	A Socket/X control.
integer	An integer that specifies a port.

Remarks

You must set the LocalAddress and LocalPort properties of the socket to appropriate values before connecting to a remote server.

Data Type

Integer

NotificationObject Property

Description

Fast notification interface object.

Syntax

object.**NotificationObject**

The syntax of the **NotificationObject** property has these parts:

Part	Description
object	A Socket/X control.

Remarks

Both the SocketX COM object and control support fast-notification callback interfaces. If you Implement ISocketXObjNotify or ISocketXCtlNotify in a form or class, you can assign an instance of that form or class to the SocketX object or control's NotificationObject.

Data Type

ISocketXCtlNotify

SocketXCtlNotify Interface Accept Event

Description

The Accept event is fired when a client connects to a listening socket.

Syntax

Sub *object_***Accept(**[*index* **As Integer**]**)**

The syntax of the **Accept** event has these parts:

Part	Description
object	Required. An instance of a SocketXCtlNotify interface.

Remarks

To establish a connection with a client, you need to have at least two sockets, one to Listen for client connections and another which is used to establish the client connection. If you have two socket controls named Server and ClientConnection, then the following code will establish a connection with a client:

```
Sub Server_Accept (SocketHandle As Long, ErrorCode As Integer)
    ClientConnection.Socket = SocketHandle
End Sub
```

If you want to support multiple clients simultaneously, you will need to use either a control array, or multiple forms, each with a socket control. See the SERVER3 sample project for an example of a multiple-client server.

Accept Event

See Also Error Codes

Description

The Accept event is fired when a client connects to a listening socket.

Syntax

Sub *object*_**Accept(**[*index* **As Integer**,] *SocketHandle* **As Integer**, *ErrorCode* **As Long)**

The syntax of the **Accept** event has these parts:

Part	Description
object	A Socket/X control.
index	An integer that identifies a control if it's in a control array.
SocketHandle	An integer that specifies the handle of the new socket.
ErrorCode	A long integer that specifies an error, if any.

Remarks

To establish a connection with a client, you need to have at least two sockets, one to Listen for client connections and another which is used to establish the client connection. If you have two socket controls named Server and ClientConnection, then the following code will establish a connection with a client:

```
Sub Server_Accept (SocketHandle As Long, ErrorCode As Integer)
    ClientConnection.Socket = SocketHandle
End Sub
```

If you want to support multiple clients simultaneously, you will need to use either a control array, or multiple forms, each with a socket control. See the SERVER3 sample project for an example of a multiple-client server.

Close Event

See Also Error Codes

Description

The OnClose event is fired when the socket is closed.

Syntax

Sub *object*_**Close(**[*index* **As Integer**,] *ErrorCode* **As Long)**

The syntax of the **Close** event has these parts:

Part	Description
object	A Socket/X control.
index	An integer that identifies a control if it's in a control array.
ErrorCode	A long integer that specifies an error, if any.

Remarks

This event fires when the socket closes. *ErrorCode* gives the most recent error on the closed socket.

SocketXCtlNotify Interface Close Event

Description

The OnClose event is fired when the socket is closed.

Syntax

Sub *object*_**Close(**[*index* **As Integer**]**)**

The syntax of the **Close** event has these parts:

Part	Description
object	Required. An instance of a SocketXCtlNotify interface.

Remarks

This event fires when the socket closes. *ErrorCode* gives the most recent error on the closed socket.

SocketXCtlNotify Interface Connect Event

Description

Fires when a connection is complete.

Syntax

Sub *object*_**Connect(**[*index* **As Integer**]**)**

The syntax of the **Connect** event has these parts:

Part	Description
object	Required. An instance of a SocketXCtlNotify interface.

Remarks

The Connect event is fired when the Connect method finishes connecting to a server socket.

Connect Event

See Also Error Codes

Description

Fires when a connection is complete.

Syntax

Sub *object*_**Connect(**[*index* **As Integer**,] *ErrorCode* **As Long)**

The syntax of the **Connect** event has these parts:

Part	Description
object	A Socket/X control.
index	An integer that identifies a control if it's in a control array.
ErrorCode	A long integer that specifies an error, if any.

Remarks

The Connect event is fired when the Connect method finishes connecting to a server socket.

SocketXCtlNotify Interface OutOfBandData Event

Description

Fired when out-of-band data is received.

Syntax

Sub *object*_**OutOfBandData(**[*index* **As Integer**]**)**

The syntax of the **OutOfBandData** event has these parts:

Part	Description
object	Required. An instance of a SocketXCtlNotify interface.

Remarks

Out-of-band data is a logically independent channel that is associated with each pair of connected stream sockets. Datagram sockets do not support out-of-band-data. The channel is generally used to send urgent data. To retrieve the data, use the ReceiveFrom method.

OutOfBandData Event

See Also Error Codes

Description

Fired when out-of-band data is received.

Syntax

Sub *object*_**OutOfBandData(**[*index* **As Integer**,] *ErrorCode* **As Long)**

The syntax of the **OutOfBandData** event has these parts:

Part	Description
object	A Socket/X control.
index	An integer that identifies a control if it's in a control array.
ErrorCode	A long integer that specifies an error, if any.

Remarks

Out-of-band data is a logically independent channel that is associated with each pair of connected stream sockets. Datagram sockets do not support out-of-band-data. The channel is generally used to send urgent data. To retrieve the data, use the ReceiveFrom method.

SocketXCtlNotify Interface Receive Event

Description

Fired when the socket has data available to be read.

Syntax

Sub *object*_**Receive(**[*index* **As Integer**]**)**

The syntax of the **Receive** event has these parts:

Part	Description
object	Required. An instance of a SocketXCtlNotify interface.

Remarks

When data arrives at a socket, the Receive event is fired.

The data can be read using the Receive method as in the following code:

```
Sub AsyncSocket1_Receive (ErrorCode As Integer)
    Dim t As String
    '
    ' Receive data available, get it
    '
    SocketX1.Receive
    t = SocketX1.ReceiveBuffer
    '
    ' Echo to client
    '
    SocketX1.SendBuffer = t
    SocketX1.Send
End Sub
```

Receive Event

See Also Error Codes

Description

Fired when the socket has data available to be read.

Syntax

Sub *object*_**Receive(**[*index* **As Integer**,] *ErrorCode* **As Long)**

The syntax of the **Receive** event has these parts:

Part	Description
object	A Socket/X control.
index	An integer that identifies a control if it's in a control array.
ErrorCode	A long integer that specifies an error, if any.

Remarks

When data arrives at a socket, the Receive event is fired.

The data can be read using the Receive method as in the following code:

```
Sub AsyncSocket1_Receive (ErrorCode As Integer)
    Dim t As String
    '
    ' Receive data available, get it
    '
    SocketX1.Receive
    t = SocketX1.ReceiveBuffer
    '
    ' Echo to client
    '
    SocketX1.SendBuffer = t
    SocketX1.Send
End Sub
```

Send Event

See Also Error Codes

Description

The Send event is fired when the socket connection is ready to send data.

Syntax

Sub *object*_**Send(**[*index* **As Integer**,] *ErrorCode* **As Long)**

The syntax of the **Send** event has these parts:

Part	Description
object	A Socket/X control.
index	An integer that identifies a control if it's in a control array.
ErrorCode	A long integer that specifies an error, if any.

Remarks

The Send event is fired once after the socket is connected. It is not fired each time you send data. After the Send event is received, you can send as many packets as you like, or until an error occurs, without waiting for another Send event.

Note that if the output queue becomes full (this may happen if you're sending lots of data), an error will be returned from the Send method. Then, when the socket is ready to send data again, the Send event will be fired.

One way to think of the Send event is that it is fired whenever the socket is ready for you to send data after some period of time when it was not ready. When a socket is first connected, it is not ready for you to send data. When it becomes ready, the Send event is fired. Then, if later sends overflow the output queue, the Send event will be fired after the socket once again becomes ready for data.

SocketXCtlNotify Interface Send Event

Description

The Send event is fired when the socket connection is ready to send data.

Syntax

Sub *object*_**Send(**[*index* **As Integer**]**)**

The syntax of the **Send** event has these parts:

Part	Description
object	Required. An instance of a SocketXCtlNotify interface.

Remarks

The Send event is fired once after the socket is connected. It is not fired each time you send data. After the Send event is received, you can send as many packets as you like, or until an error occurs, without waiting for another Send event.

Note that if the output queue becomes full (this may happen if you're sending lots of data), an error will be returned from the Send method. Then, when the socket is ready to send data again, the Send event will be fired.

One way to think of the Send event is that it is fired whenever the socket is ready for you to send data after some period of time when it was not ready. When a socket is first connected, it is not ready for you to send data. When it becomes ready, the Send event is fired. Then, if later sends overflow the output queue, the Send event will be fired after the socket once again becomes ready for data.

OutOfBandEnabled Property

See Also Error Codes

Description

Enables the receipt of out-of-band data on a stream socket.

Syntax

object.**OutOfBandEnabled** [= *boolean*]

The syntax of the **OutOfBandEnabled** property has these parts:

Part	Description
object	A Socket/X control.
boolean	A boolean expression that determines whether or not out of band data is enabled.

Remarks

Each pair of connected stream sockets (a client and a server, for instance) has a logically independent channel over which out-of-band data can be transmitted. This channel provides for unbuffered transmission of one packet of data which is typically of an urgent nature. Setting the OutOfBandEnabled flag to True enables the socket to receive these messages.

Data Type

Boolean

ReceiveBufferSize Property

See Also Error Codes

Description

Sets size of the socket's receive buffer.

Syntax

object.**ReceiveBufferSize** [= *integer*]

The syntax of the **ReceiveBufferSize** property has these parts:

Part	Description
object	A Socket/X control.
integer	An integer that specifies the receive buffer size.

Remarks

Used to change the default receive buffer size. The default receive buffer size should be adequate for most purposes.

The control receives bytes by accepting data from WinSock into the control's ReceiveBuffer property. However, the ReceiveBufferSize and the ReceiveBuffer are not related. When data comes into a socket, the Receive event is fired to indicate that data has arrived. When the Receive method is invoked, the data is moved from WinSock into the control's ReceiveBuffer.

The ReceiveBuffer property is a string and a 16-bit string can hold 32Kb while a 32-bit string can hold 4 gigabytes of data. The ReceiveBufferSize indicates how much data *WinSock* will buffer before the Receive method is invoked. If the ReceiveBufferSize is 2K and three 1K packets arrive before the Receive method is invoked, data would be lost.

Note: this property specifies the size of the buffer used internally by WinSock and has nothing to do with the ReceiveBuffer property which will simply hold as much data as was received by any read.

Data Type

Integer

Receive Method

See Also Error Codes

Description

Used to retrieve receive data from the socket control.

Syntax

object.**Receive**

The syntax of the **Receive** method has these parts:

Part	Description
object	Required. A Socket/X control.

Remarks

This method receives data from the current socket.

Example:

```
str = AsyncSocket1.Receive()
```

ReceiveBuffer Property

See Also Error Codes

Description

Contains received data after the Receive method is invoked.

Syntax

object.**ReceiveBuffer** [= *string*]

The syntax of the **ReceiveBuffer** property has these parts:

Part	Description
object	A Socket/X control.
string	A string expression that contains the data received.

Remarks

Data received from a remote connected socket is accessed through the ReceiveBuffer property.

Data Type

String

ReceiveFrom Method

See Also Error Codes

Description

Receives a datagram and stores the source address.

Syntax

object.**ReceiveFrom***RemoteAddress*

The syntax of the **ReceiveFrom** method has these parts:

Part	Description
object	Required. A Socket/X control.
RemoteAddress	Optional. A string expression that specifies the address of the socket that is sending data.

Remarks

When reading from a datagram socket, or when reading out-of-band-data, you should use the ReceiveForm method. The data is stored in the ReceiveBuffer and the sending socket's address and port number are returned in the RemoteAddress and RemotePort and in the corresponding control properties.

When reading out-of-band data from a stream socket, you must specify the soxOOBEvent flag, and with either datagram or stream sockets you can remove the receive data from the incoming queue by passing the ASocketMsgPeek flag.

ReceiveTimeout Property

See Also Error Codes

Description

Specifies the number of seconds to wait to receive data.

Syntax

object.**ReceiveTimeout** [= *integer*]

The syntax of the **ReceiveTimeout** property has these parts:

Part	Description
object	A Socket/X control.
integer	An integer that specifies a duration, in seconds.

Remarks

If you set this property to a non-zero value then, after that number of seconds has elapsed while waiting for data, an error will be returned.

Data Type

Integer

RemoteAddress Property

See Also Error Codes

Description

Address of remote socket (i.e., 123.123.12.3).

Syntax

object.**RemoteAddress** [= *string*]

The syntax of the **RemoteAddress** property has these parts:

Part	Description
object	A Socket/X control.
string	A string expression that specifies an IP address.

Remarks

RemoteAddress and RemotePort are used prior to invoking the Connect method. These two properties indicate the remote socket (remote IP and remote port) to connect with.

Data Type

String

RemoteName Property

See Also Error Codes

Description

Name of remote socket (i.e., ftp.microsoft.com).

Syntax

object.**RemoteName** [= *string*]

The syntax of the **RemoteName** property has these parts:

Part	Description
object	A Socket/X control.
string	A string expression that specifies a host name.

Remarks

When the RemoteNameAddrXlate property is set to True, setting the RemoteName property will attempt to determine the corresponding IP address and set the RemoteAddress property accordingly.

Note that, with WinSock, it is not possible to do this both quickly and reliably. The control performs the lookup in such a way that it will not hang for 2-3 minutes if the computer is not connected to the net. However, the trade-off for this is that it may happen occasionally that the lookup fails, even though the network is actually accessible. For completely reliable name lookups, you should use the GetHst control.

Data Type

String

RemotePort Property

See Also Error Codes

Description

Specifies the port address of the remote socket.

Syntax

object.**RemotePort** [= *integer*]

The syntax of the **RemotePort** property has these parts:

Part	Description
object	A Socket/X control.
integer	An integer that specifies a port.

Remarks

Together RemotePort and RemoteAddress fully specify a remote socket's address.

Data Type

Integer

RemoteNameAddrXlate Property

See Also Error Codes

Description

Used to translate names, such as "mabry.com", to IP addresses (204.157.98.11).

Syntax

object.**RemoteNameAddrXlate** [= *boolean*]

The syntax of the **RemoteNameAddrXlate** property has these parts:

Part	Description
object	A Socket/X control.
boolean	A boolean expression that indicates whether host names will be translated into IP addresses.

Remarks

When RemoteNameAddrXlate is True, changes to either RemoteName or RemoteAddress will initiate an attempt to use a DNS (domain name server) to translate one to the other. For instance, if RemoteName is set to "activexpert.com", RemoteNameAddrXlate is True, and if the local machine is connected to the TCP/IP network, the control will attempt to translate "activexpert.com" to its IP address "206.161.236.182".

If the attempted translation fails (between either RemoteAddress or RemoteName) because the local machine is not attached to the TCP/IP network, or it cannot reach its DNS, the other property is left unchanged. If an error occurs for any other reason, the other property is set to an empty string.

The following sequence can be used to obtain a host's address:

```
AsyncSocket1.RemoteNameAddrXlate = FALSE
AsyncSocket1.RemoteAddress = "0.0.0.0"
AsyncSocket1.RemoteNameAddrXlate = TRUE
AsyncSocket1.RemoteName = "activexpert.com"
If (AsyncSocket1.RemoteAddress <> "" ) Then
    Debug.Print AsyncSocket1.RemoteAddress
EndIf
```

Please see the Remarks section of the RemoteName property for further comments on the use of this property.

Data Type

Boolean

ReuseAddressEnabled Property

See Also Error Codes

Description

Enables other sockets to use the same address as this socket.

Syntax

object.**ReuseAddressEnabled** [= *boolean*]

The syntax of the **ReuseAddressEnabled** property has these parts:

Part	Description
object	A Socket/X control.
boolean	A boolean expression that indicates whether socket addresses may be re-used.

Remarks

By default, a socket cannot use a local address which is already in use. On occasion, however, it may be desirable to reuse an address in this way. Since every connection is uniquely identified by the combination of local and remote addresses, there is no problem with having two sockets use the same local address as long as the remote addresses are different.

The ReuseAddressEnabled property must be set to True before invoking a method. Otherwise, it has no effect.

Data Type

Boolean

RouteEnabled Property

See Also Error Codes

Description

Enables packet routing.

Syntax

object.**RouteEnabled** [= *boolean*]

The syntax of the **RouteEnabled** property has these parts:

Part	Description
object	A Socket/X control.
boolean	A boolean expression that indicates whether packet routing is enabled.

Remarks

When this property is set to True, packet routing is enabled.

Data Type

Boolean

Send Method

See Also Error Codes

Description

Sends a buffer of data to the connected socket.

Syntax

object.**Send***SendBuffer*

The syntax of the **Send** method has these parts:

Part	Description
object	Required. A Socket/X control.
SendBuffer	Optional. A string expression that specifies the data to send. If not specified, the SendBuffer property is sent.

Remarks

Sends the contents of the SendBuffer property to the connected socket. When using the ActiveX / OCX, you can supply *SendBuffer* as an argument and it will be sent instead of the contents of the SendBuffer property.

There are two special situations to watch out for when sending data. Both are signaled by firing an error during execution of the Send method. For this reason, you will need to be sure to have error handling code in place where your code sends data over a socket.

No matter what protocol stack is in use on a machine, there will be a limit as to how much data may be queued for transmission at any one time. When the output queue is full (or nearly full) and you call the Send method, either a 10035 (WSAEWOULDBLOCK) error will be fired, or a 20000 (could not send all data) error will be fired. The first error (10035) tells you that none of the data was queued. The second error (20000) tells you that some of the data was queued. The BytesSent property can then be examined to determine how much data was sent. In either case, you must wait until the next Send event occurs before continuing to send data over the socket.

SendBuffer Property

See Also Error Codes

Description

Buffer for data to be sent to the remote socket.

Syntax

object.**SendBuffer** [= *string*]

The syntax of the **SendBuffer** property has these parts:

Part	Description
object	A Socket/X control.
string	A string expression that contains the data to send.

Remarks

Data to send to connected socket.

Data Type

String

SendBufferSize Property

See Also Error Codes

Description

Sets the size of the socket's send buffer.

Syntax

object.**SendBufferSize** [= *integer*]

The syntax of the **SendBufferSize** property has these parts:

Part	Description
object	A Socket/X control.
integer	An integer that specifies the send buffer size.

Remarks

Used to change the default transmit buffer size.

Note: this property specifies the size of the buffer used internally by Winsock and has nothing to do with the SendBuffer property.

Data Type

Integer

SendTimeout Property

See Also Error Codes

Description

Specifies the number of seconds to wait to send data.

Syntax

object.**SendTimeout** [= *integer*]

The syntax of the **SendTimeout** property has these parts:

Part	Description
object	A Socket/X control.
integer	An integer that specifies a duration, in seconds.

Remarks

If you set this property to a non-zero value, after that number of seconds has elapsed while waiting to send data, an error will be returned.

Data Type

Integer

SendTo Method

See Also Error Codes

Description

Sends data to a specific remote socket.

Syntax

object.**SendTo***RemoteAddress*,*SendBuffer*

The syntax of the **SendTo** method has these parts:

Part	Description
object	Required. A Socket/X control.
RemoteAddress	Optional. A string expression that specifies the address of the socket to send to.
SendBuffer	Optional. A string expression that specifies the data to send. If not specified, the SendBuffer property is sent.

Remarks

SendTo is used to send the contents of *SendBuffer* (using a socket of type soxDatagram) to another socket as specified by the RemoteAddress and RemotePort properties.

Socket Property

See Also Error Codes

Description

Windows socket handle.

Syntax

object.**Socket** [= *long*]

The syntax of the **Socket** property has these parts:

Part	Description
object	A Socket/X control.
long	A long integer that specifies a socket handle.

Remarks

This property exists solely for the purpose of accepting a client connection request. The Accept event passes a Socket handle as one of its parameters. That parameter must be assigned to the Socket property of some control other than the one firing the Accept event, in order to accept the connection.

Data Type

Long

SocketAddress Property

See Also Error Codes

Description

The local socket's IP address.

Syntax

object.**SocketAddress** [= *string*]

The syntax of the **SocketAddress** property has these parts:

Part	Description
object	A Socket/X control.
string	A string expression that specifies an IP address.

Remarks

This property is especially useful when connecting to the Internet through an ISP which dynamically assigns IP addresses. After a remote socket connection has been established, this property will return the IP address assigned to the socket. This property is also useful when a Connect call has been made without doing a Bind first. This provides the only means by which you can determine the local association which has been set by the system.

Windows keeps track of files using file handles and window objects using window handles (hWnd). Likewise, Windows keeps track of sockets using socket handles. The Socket property is a long integer that contains the socket handle for a given socket. When a client connects to a listening server, the listening control use the socket handle tp effectively pass the connection over to another control to handle the communication so the listening socket can return to listening for additional client connection requests. When the socket its passed from one control to another via the socket handle, the SocketAddress and SocketPort property indicate the IP address and the port of the socket.

In other words, we use the LocalAddress and LocalPort when explicitly creating a socket. When a socket is passed from a listening control to a communication control, the SocketAddress and SocketPort contain the IP and port of the socket being passed. The SocketAddress of the communication control will usually be the same as the LocalAddress of the listening control since they will be on the same machine, but it may be possible for the two to be different on a mult-homed machine (ie -- a single machine that has multiple IPs bound to the same NIC).

Data Type

String

SocketPort Property

See Also Error Codes

Description

Local port number.

Syntax

object.**SocketPort** [= *integer*]

The syntax of the **SocketPort** property has these parts:

Part	Description
object	A Socket/X control.
integer	An integer that specifies a port.

Remarks

Port number used by the socket. This property is valid only after a remote system is connected.

Data Type

Integer

SocketType Property

See Also Error Codes

Description

Determines whether a socket will be a stream or datagram socket.

Syntax

object.**SocketType** [= *integer*]

The syntax of the **SocketType** property has these parts:

Part	Description
object	A Socket/X control.
integer	An integer that determines the style of socket: stream or datagram.

Remarks

The SocketType property may be one of the following:

Constant	Value	Description
soxStream	0	Provides sequenced, reliable, full-duplex, connection-based byte streams.
soxDatagram	1	Supports datagrams, which are connectionless, unreliable packets of a fixed (typically small) maximum length.

Data Type

Integer

SocketXCtlNotify Interface

Events

Description

Fast Notification interface.

Remarks

The SocketX COM Object and OCX also implement a Fast Notification (early-bound callback) interface as an alternative to using events. If you add the following line to a form (or control, or class module):

```
Implements ISocketXObjNotify
```

You will then get to implement the early-bound callback functions which are identical to the events except that an instance of the COM Object (or OCX) calling the function is passed as the first parameter. This extra parameter substitutes for the Index parameter you would have had in a control array implementation.

State Property

See Also

Description

The current state of the control.

Syntax

object.**State**

The syntax of the **State** property has these parts:

Part	Description
object	A Socket/X control.

Remarks

The following constants represent the set of values that are assigned to the State property:

Constant	Value	Description
soxUnusable	0	The socket is unusable. This occurs when the socket stack specified by LibraryName cannot be loaded.
soxClosed	1	The socket is usable, but not yet created.
soxBound	2	The socket has been bound to a local address.
soxCreated	3	The socket has been created. A socket must be created before using the Bind or Connect methods.
soxConnected	4	The socket has been connected. State is set to soxConnected when a connection is established using the Connect method or after a socket has accepted a client connection by assigning its Socket property within an Accept event.

Data Type

Integer

StateString Property

See Also

Description

A user-friendly version of the current state of the control.

Syntax

object.**StateString**

The syntax of the **StateString** property has these parts:

Part	Description
object	A Socket/X control.

Remarks

This property returns a string expression that tells the current state of the control.

This property is read-only and only available at run-time.

Data Type

String

TCPNoDelayEnabled Property

See Also Error Codes

Description

Disables the Nagle delay algorithm when set to True.

Syntax

object.**TCPNoDelayEnabled** [= *boolean*]

The syntax of the **TCPNoDelayEnabled** property has these parts:

Part	Description
object	A Socket/X control.
boolean	A boolean expression that indicates whether the Nagle delay algorithm is disabled.

Remarks

The NoDelayEnabled property enables the Nagle algorithm. The Nagle algorithm is used to reduce the number of small packets sent by a host by buffering unacknowledged send data until a full-size packet can be sent. However, for some applications this algorithm can impede performance, and NoDelayEnabled be used to turn it off. You should not set this property to True unless the impact of doing so is well-understood and desired.

Data Type

Boolean

Version Property

Description

Returns the version of the control.

Syntax

object.**Version**

The syntax of the **Version** property has these parts:

Part	Description
object	A Socket/X control.

Remarks

This property holds the current version of the control. It is read-only and available at both design-time and run-time.

Data Type

String

WSAVersion Property

Description

Socket stack version.

Syntax

object.**WSAVersion**

The syntax of the **WSAVersion** property has these parts:

Part	Description
object	A Socket/X control.

Remarks

The version of the Windows Sockets specification that the Windows Sockets DLL uses.

Data Type

String

WSDescription Property

Description

The socket stack's description of itself.

Syntax

object.**WSDescription**

The syntax of the **WSDescription** property has these parts:

Part	Description
object	A Socket/X control.

Remarks

Socket stacks may provide a string containing some text description. If the socket stack specified by LibraryName provides a description string, this property provides it to you.

Data Type

String

WSMaxSockets Property

Description

Maximum number of sockets supported by the current socket stack.

Syntax

object.**WSMaxSockets**

The syntax of the **WSMaxSockets** property has these parts:

Part	Description
object	A Socket/X control.

Remarks

The maximum number of sockets which a single process can potentially open. A Windows Sockets implementation can provide a global pool of sockets for allocation to any process. Alternatively, it can allocate per-process resources for sockets. The number can well reflect the way in which the Windows Sockets DLL or the networking software was configured. Application writers can use this number as a crude indication of whether the Windows Sockets implementation is usable by the application. There is no guarantee that a particular application can actually allocate WSMaxSockets sockets, since there can be other Windows Sockets applications in use.

Note: Not all socket stacks provide a useful value for the WSAMaxSockets stack. Microsoft's 32-bit stack, for instance, reports zero as the maximum number of available sockets.

Data Type

Integer

WSMaxUDPSize Property

Description

Maximum datagram size.

Syntax

object.**WSMaxUDPSize**

The syntax of the **WSMaxUDPSize** property has these parts:

Part	Description
object	A Socket/X control.

Remarks

The size in bytes of the largest User Datagram Protocol (UDP) datagram that can be sent or received by a Windows Sockets application. If the implementation imposes no limit, WSAMaxUdpSize is zero. In many implementations of Berkeley sockets, there is an implicit limit of 8192 bytes on UDP datagrams (which are fragmented if necessary). A Windows Sockets implementation can impose a limit based, for instance, on the allocation of fragment reassembly buffers. The minimum value of WSAMaxUdpSize for a compliant Windows Sockets implementation is 512.

Note that regardless of the value of WSAMaxUdpSize, it is inadvisable to attempt to send a broadcast datagram which is larger than the Maximum Transmission Unit (MTU) for the network. (The Windows Sockets API does not provide a mechanism to discover the MTU, but it must be no less than 512 bytes.)

Data Type

Long

Getting Custom Controls Written

If you or your organization would like to have custom controls written, you can contact us at the following:

Mabry Software, Inc.
503 316th Street Northwest
Stanwood, WA 98292

Phone: 360-629-9278
Fax: 360-629-9278

Internet: mabry@mabry.com

You can also contact Zane Thomas. He can be reached at:

Zane Thomas
Post Office Box 121
Indianola, WA 98342

Internet: zane@mabry.com

Licensing Information

Legalese Version

Mabry Software grants a license to use the enclosed software to the original purchaser. Copies may be made for back-up purposes only. Copies made for any other purpose are expressly prohibited, and adherence to this requirement is the sole responsibility of the purchaser.

Customer written executable applications containing embedded Mabry products may be freely distributed, without royalty payments to Mabry Software, provided that such distributed Mabry product is bound into these applications in such a way so as to prohibit separate use in design mode, and that such Mabry product is distributed only in conjunction with the customers own software product. The Mabry Software product may not be distributed by itself in any form.

Neither source code for Mabry Software products nor modified source code for Mabry Software products may be distributed under any circumstances, nor may you distribute .OBJ, .LIB, etc. files that contain our routines. This control may be used as a constituent control only if the compound control thus created is distributed with and as an integral part of an application. Permission to use this control as a constituent control does not grant a right to distribute the license (LIC) file or any other file other than the control executable itself. This license may be transferred to a third party only if all existing copies of the software and its documentation are also transferred.

This product is licensed for use by only one developer at a time. Mabry Software expressly prohibits installing this product on more than one computer if there is any chance that both copies will be used simultaneously. This restriction also extends to installation on a network server, if more than one workstation will be accessing the product. All developers working on a project which includes a Mabry Software product, even though not working directly with the Mabry product, are required to purchase a license for that Mabry product.

This software is provided as is. Mabry Software makes no warranty, expressed or implied, with regard to the software. All implied warranties, including the warranties of merchantability and fitness for a particular use, are hereby excluded.

MABRY SOFTWARE'S LIABILITY IS LIMITED TO THE PURCHASE PRICE. Under no circumstances shall Mabry Software or the authors of this product be liable for any incidental or consequential damages, nor for any damages in excess of the original purchase price.

To be eligible for free technical support by telephone, the Internet, CompuServe, etc. and to ensure that you are notified of any future updates, please complete the enclosed registration card and return it to Mabry Software.

English Version

We require that you purchase one copy of a control per developer on a project. If this is met, you may distribute the control with your application royalty free. You may never distribute the LIC file. You may not change the product in any way that removes or changes the requirement of a license file.

We encourage the use of our controls as constituent controls when the compound controls you create are an integral part of your application. But we don't allow distribution of our controls as constituents of other controls when the compound control is not part of an application. The reason we need to have this restriction is that without it someone might decide to use our control as a constituent, add some trivial (or even non-trivial) enhancements and then sell the compound control. Obviously there would be little difference between that and just plain reselling our control.

If you have purchased the source code, you may not re-distribute the source code either (nor may you copy it into your own project). Mabry Software retains the copyright to the source code.

Your license is transferable. The original purchaser of the product must make the transfer request. Contact us for further information.

The sample versions of our products are intended for evaluation purposes only. You may not use the sample version to develop completed applications.