

# **Windows Sockets ActiveX Control**

**TCP/IP Windows Sockets  
ActiveX Control  
for Microsoft® Windows™**

**Version 5.2**

**Copyright © 1995 - 2003 by Distinct® Corporation  
All rights reserved**

**Distinct Corporation**

3315 Almaden Expressway  
San Jose, CA 95118 USA

Phone: +1 408-445-3270

Fax: +1 408-445-3274

Email: [sales@distinct.com](mailto:sales@distinct.com)

WWW: <http://www.distinct.com>

**Disclaimer**

Distinct Corporation makes no warranties as to the contents of this documentation and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Information in this manual is subject to change without notice and does not represent a commitment on the part of Distinct Corporation. The Software described in this manual is furnished under a license agreement and may be used or copied only in accordance with the terms of that agreement.

**Copyright Notice**

© 1995 - 2003 by Distinct Corporation. All rights reserved.

No part of this publication may be reproduced, transmitted or translated into any language by any means without the express written permission of Distinct Corporation.

**Trademarks**

Distinct is a registered trademark and Visual Internet Toolkit is a trademark of Distinct Corporation. Windows is a registered trademark of Microsoft Corporation. Other product names are trademarks or registered trademarks of their respective owners.

Last updated June 5th, 2003

Published in the United States of America



## **Table of Contents**

Table of Contents .....	4
<b>1 Windows Sockets ActiveX Overview</b> .....	7
1.1 Introduction .....	7
1.2 Usage of the Windows Sockets ActiveX .....	7
1.3 Windows Sockets ActiveX Property Summary .....	9
1.4 Windows Sockets ActiveX Event Summary .....	11
1.5 Windows Sockets ActiveX Method Summary .....	12
1.6 D_SKT.TXT .....	13
<b>2 Windows Sockets Properties</b> .....	17
2.1 Action .....	17
2.2 BinaryData .....	18
2.3 Convert .....	20
2.4 Error .....	22
2.5 FirewallPort .....	29
2.6 FirewallServer .....	30
2.7 FwAddrType .....	31
2.8 FwAuthMethods .....	32
2.9 FwPassword .....	33
2.10 FwSocksVer .....	34
2.11 FwUsername .....	35
2.12 HostAddress .....	36
2.13 HostConvert .....	37
2.14 HostName .....	38
2.15 Local .....	39
2.16 LocalHostAddress .....	40
2.17 LocalPort .....	41
2.18 Option .....	42
2.19 OptionValue .....	44
2.20 Protocol .....	45
2.21 ProtocolName .....	46
2.22 ProtocolNumber .....	47
2.23 Receive .....	48
2.24 ReceiveCount .....	49
2.25 ReceiveLen .....	50
2.26 ReceiveOOB .....	51
2.27 ReceiveSize .....	52
2.28 RemotePort .....	53
2.29 Result .....	54
2.30 Send .....	55
2.31 SendOOB .....	56
2.32 SendSize .....	57
2.33 SendTimeout .....	58
2.34 ServiceName .....	59
2.35 ServicePort .....	60
2.36 SourceAddress .....	61
2.37 SourcePort .....	62
<b>3 Windows Sockets Events</b> .....	63
3.1 OnAccept .....	63
3.2 OnClose .....	64
3.3 OnConnect .....	65
3.4 OnError .....	66

3.5	OnReceive	71
3.6	OnReceiveOOB	72
4	Windows Sockets Methods	73
4.1	Accept	73
4.2	AddrToDecimal	74
4.3	AddrToName	75
4.4	AsyncConnect	76
4.5	Broadcast	77
4.6	Connect	78
4.7	Disconnect	79
4.8	FwConnect	80
4.9	KeepAlive	82
4.10	Linger	83
4.11	Listen	84
4.12	NameToAddr	85
4.13	ProtoNameToNumber	86
4.14	ProtoNumberToName	87
4.15	ReceiveB	88
4.16	RecvOOB	90
4.17	ReuseAddr	91
4.18	SendB	92
4.19	SvcNameToPort	94
4.20	SvcPortToName	95



## 1 Windows Sockets ActiveX Overview

### 1.1 Introduction

The Distinct Windows Socket ActiveX provides your application with a fast, robust, secure interface for sending and receiving data between computers. You can create Windows Sockets applications that run over TCP or UDP protocols without ever making a single socket call. For ease of use all short and long integer values used (such as host addresses and ports) are specified in local host byte order. This frees you from converting arguments back and forth between function calls that require different byte orders for their arguments. Built-in firewall proxy support is available for TCP connections.

### 1.2 Usage of the Windows Sockets ActiveX

See the section entitled "Using Distinct ActiveX controls in various environments" on how to add the control to your project.

After placing a Windows Socket ActiveX control into a form, some properties can be set at design time. If the ActiveX control will always establish the same type of connection, then the Protocol property (TCP or UDP) and the LocalPort property can be preset. If the connection will always be made to the same remote system, then the HostName or HostAddress and the RemotePort properties can also be defined at design time.

Several properties control the way data is buffered during send and receive operations and how it is transferred to the application during a read operation. The SendSize and ReceiveSize properties specify the size of the send and receive buffers. Applications which will send large amounts of data can increase the size of the send buffer and applications which will receive large amounts of data can increase the size of the receive buffer. By increasing send or receive buffer sizes, the ActiveX control may be able to increase throughput for some applications. The two buffer sizes are independent of each other.

The ReceiveLen property controls the maximum number of bytes of data which are read every time the receive buffer is accessed with the Receive property. The default value of 100 bytes should be sufficient for most applications. The SendTimeout property specifies the time-out value in seconds for sending data in the send buffer to the remote machine. The Windows Socket ActiveX control will wait for the specified period of time (default is 20 seconds) to transmit the data in the send buffer. If the underlying protocol stack is unable to deliver the data within the given time, then the ActiveX control will time-out and return an error.

Properties which are set at design time can still be altered at run time.

The following properties can only be accessed at run time. The Action property is used to establish or close a connection and to listen for incoming connection requests. The HostConvert property converts host names to their corresponding internet addresses and vice versa. Since this operation may involve contacting a name server over the network, it can only be invoked at run time. The Receive and ReceiveCount properties are used to access data which has been received from the remote system over an established connection. The Send property sends data to the remote system over an established connection. The Result property contains the number of bytes sent to the remote machine or received from the remote machine. The value of this property can be checked after each send operation to ensure that all of the data was sent and after each receive operation to determine the number of bytes read from the receive buffer.

TCP connections can be established in active (client) or passive (server) mode, but UDP connections are peer to peer and therefore do not use the client and server model. Connections for TCP clients and UDP can be established by setting the Action property to ACTION\_CONNECT (or by calling the Connect method). Before connecting, the Protocol, LocalPort (default value of 0 can be used), HostAddress and RemotePort properties must be initialized properly. If the connection can be established, the OnConnect event will occur before the next line of code is reached. TCP connections can also be established in asynchronous mode. If the Option property is set to OPTION\_ASYNC\_CONNECT\_ON (default is off) or if the AsyncConnect method is used, then setting the Action property to ACTION\_CONNECT (or the Connect method) will not wait for the connection to be established. Instead, the OnConnect event will occur later once the connection has been established (this may take several seconds).

For some users, the local machine is located on a different subnet than the remote host and the only way of communication is through a firewall. If this is the case, then the built-in firewall support of the Windows Socket ActiveX control can be used to establish a session via a firewall by setting the Action property to ACTION\_FW\_CONNECT (or by calling the FwConnect method). In addition to setting the properties for a regular session, the FirewallServer, FirewallPort, FwAddrType, FwAuthMethods, FwUsername and FwPassword properties must also be set before setting the Action property to ACTION\_FW\_CONNECT. If the connection can be established, the OnConnect event will occur before the next line of code is reached.

Connections for TCP servers can be established by first setting the socket to listening mode by setting the Action property to ACTION\_LISTEN (or by calling the Listen method). Before listening, the Protocol (always TCP) and LocalPort (default value of 0 cannot be used) properties must be initialized properly. Whenever a client sends a connection request over the network to the port on which the socket is listening, an OnAccept event is fired for the server TCP control. To establish a connection with the client, the Action property must be set to ACTION\_ACCEPT (or call the Accept method) in response to this event. If the connection can be established, the OnConnect event will occur before the next line of code is reached.

The Option property can be used to set or reset specific socket options which change the behavior of the Windows Socket ActiveX control. These options affect socket operations, such as whether broadcast messages may be sent on the socket, whether out-of-band data can be received and whether a socket may be bound to a local address which is already in use. The Option property can also control the actions taken when unsent data is queued in the send buffer when a connection is closed and can specify if the underlying protocol stack should use "keep-alive" packets on TCP connections. More details are given on the reference pages of the Option and OptionValue properties.

Once a client or server connection has been established, data can be sent with the Send property and available data can be received with the Receive property. Every time data is read from the receive buffer, the SourceAddress and SourcePort properties can be consulted to determine from which machine and port the data originated. This source information will not change while connected with another machine over TCP, but can change from one read request to another for UDP. TCP connections are established with another machine for the entire length of the session, whereas UDP sockets can send and receive data to and from different hosts (or even broadcast) with each send and receive operation. Binary data can also be transferred using the ReceiveB and SendB methods.

Out-of-band data is generally received separately from normal data. However, out-of-band data can also be received over the normal data stream by setting this socket option using the Option property. Out-of-band data can be sent and received using the SendOOB and ReceiveOOB properties. Arrival of out-of-band data will cause OnReceiveOOB events.

The Error property contains the error code returned by the underlying protocol stack if a socket operation could not be completed successfully. This value can be used to determine the exact nature of an error.

Once a connection is no longer needed or no more connection requests will be accepted on a listening socket, the Action property must be set to ACTION\_DISCONNECT (or call the Disconnect method). Once the socket is closed, the OnClose event will occur before the next line of code is reached. Applications must close any connected or listening socket in this way before terminating.

### 1.3 Windows Sockets ActiveX Property Summary

**Action**

Connect, listen, accept or disconnect on a socket

**BinaryData**

Contains the binary data received following a call to the RetrieveB method.

**Convert**

Convert protocol name to number and vice versa or service name to service port and vice versa

**Error**

Error code returned by underlying protocol stack

**FirewallPort**

Firewall server port

**FirewallServer**

Name or dotted decimal internet address of firewall server

**FwAddrType**

The address type of the destination host

**FwAuthMethods**

Firewall authentication methods

**FwUsername**

Firewall username

**FwPassword**

Firewall password

**FwSocksVer**

The firewall server version

**HostAddress**

Resolved internet address of host

**HostConvert**

Convert host name to address or address to host name or address to dotted decimal

**HostName**

Name of host or dotted decimal internet address

**Local**

Name or dotted decimal internet address of local host

**LocalHostAddress**

Internet address of local host to a connection

**LocalPort**

Local port to be used for next connection

**Option**

Socket option

**OptionValue**

Socket option value

**Protocol**

Protocol to be used for next connection

**ProtocolName**

Official protocol name

**ProtocolNumber**

Official protocol number

**Receive**

Receive buffer

**ReceiveCount**

Number of bytes waiting in receive buffer

**ReceiveLen**

Maximum number of bytes to read from receive buffer

**ReceiveOOB**

Receive buffer for out-of-band data

**ReceiveSize**

Size in bytes of receive buffer

**RemotePort**

Remote port to be used for next connection

**Result**

Number of bytes sent or received

**Send**

Send buffer

**SendOOB**

Send buffer for out-of-band data

**SendSize**

Size in bytes of send buffer

**SendTimeout**

Time-out when sending data

**ServiceName**

Official service name

**ServicePort**

Port number on which service is available

**SourceAddress**

Source internet address of last received datagram or connection request

**SourcePort**

Source port of last received datagram or connection request

## 1.4 Windows Sockets ActiveX Event Summary

### **OnAccept**

Connection request has been received and is ready to be accepted

### **OnClose**

Connection has been closed

### **OnConnect**

Connection has been established

### **OnError**

Local error has occurred

### **OnReceive**

More data has been added to the receive buffer

### **OnReceiveOOB**

More out-of-band data has been received

## 1.5 Windows Sockets ActiveX Method Summary

### **Accept**

Accept an incoming connection request

### **AddrToDecimal**

Convert host address to dotted decimal address

### **AddrToName**

Convert host address to host name

### **AsyncConnect**

Use or not use asynchronous connect

### **Broadcast**

Allow or not allow transmission of broadcast messages

### **Connect**

Establish connection.

### **Disconnect**

Close connection

### **FwConnect**

Establish connection via a firewall

### **KeepAlive**

Send or not send keep-alive TCP packets

### **Linger**

Set or not set linger time-out for close socket operation

### **Listen**

Listen for incoming connection request

### **NameToAddr**

Convert host name to host address

### **ProtoNameToNumber**

Convert protocol name to protocol number

### **ProtoNumberToPort**

Convert protocol number to protocol name

### **ReceiveB**

Receive binary data

### **RecvOOB**

Receive out of band data separately or as regular data

### **ReuseAddr**

Allow or not allow reuse of local address

### **SendB**

Send binary data

### **SvcNameToPort**

Convert service name to service port

### **SvcPortToName**

Convert service port to service name

## 1.6 D\_SKT.TXT

The following provides a complete listing of the D\_SKT.TXT definition file. If your application uses more than one Distinct ActiveX control in the same form, then some definitions will conflict. For example, the FTP Client ActiveX control includes the definition

```
Global Const ACTION_DISCONNECT = 3
```

in the D\_FTP.TXT file and the Telnet ActiveX control includes the definition

```
Global Const ACTION_DISCONNECT = 2
```

in the D\_TNET.TXT file. To avoid this conflict, you must rename at least one of the constants (for example, FTP\_ACTION\_DISCONNECT or TNET\_ACTION\_DISCONNECT).

```
' Windows Sockets ActiveX Control
' (C) Copyright 1995 - 1998 by Distinct Corporation
' All rights reserved

' actions
Global Const ACTION_NONE = 0
Global Const ACTION_CONNECT = 1
Global Const ACTION_LISTEN = 2
Global Const ACTION_ACCEPT = 3
Global Const ACTION_DISCONNECT = 4
Global Const ACTION_FW_CONNECT = 5

' address conversions
Global Const CONVERT_NONE = 0
Global Const CONVERT_ADDRESS_TO_NAME = 1
Global Const CONVERT_NAME_TO_ADDRESS = 2
Global Const CONVERT_ADDRESS_TO_DECIMAL = 3

' protocol and service conversions
' CONVERT_NONE is as defined above
Global Const PROTO_NAME_TO_NUMBER = 1
Global Const PROTO_NUMBER_TO_NAME = 2
Global Const SVC_NAME_TO_PORT = 3
Global Const SVC_PORT_TO_NAME = 4

' protocols
Global Const PROTOCOL_TCP = 0
Global Const PROTOCOL_UDP = 1

' options
Global Const OPTION_NONE = 0
Global Const OPTION_BROADCAST_ON = 1
Global Const OPTION_BROADCAST_OFF = 2
Global Const OPTION_LINGER_ON = 3
Global Const OPTION_LINGER_OFF = 4
Global Const OPTION_KEEPALIVE_ON = 5
Global Const OPTION_KEEPALIVE_OFF = 6
Global Const OPTION_REUSEADDR_ON = 7
Global Const OPTION_REUSEADDR_OFF = 8
Global Const OPTION_ASYNC_CONNECT_ON = 9
Global Const OPTION_ASYNC_CONNECT_OFF = 10
Global Const OPTION_RECV_OOB_ON = 11
Global Const OPTION_RECV_OOB_OFF = 12
```

```
' the address type of the destination host
Global Const FW_ADDR_IP4 = 1
Global Const FW_ADDR_DNS = 3
Global Const FW_ADDR_IP6 = 4

'the firewall server version
Global Const FW_VERSION5 = 2
Global Const FW_VERSION4 = 4

' error codes
Global Const ERR_CHANGE_PROTOCOL = 1
Global Const ERR_CHANGE_LOCAL_PORT = 2
Global Const ERR_CHANGE_REMOTE_PORT = 3
Global Const ERR_CHANGE_RECV_SIZE = 4
Global Const ERR_CHANGE_SEND_SIZE = 5
Global Const ERR_CHANGE_HOST_ADDR = 6
Global Const ERR_NO_HOST_ADDR = 7
Global Const ERR_CANNOT_GET_SOCKET = 8
Global Const ERR_CANNOT_BIND_SOCKET = 9
Global Const ERR_CANNOT_CONNECT_SOCKET = 10
Global Const ERR_CANNOT_SETUP_SOCKET = 11
Global Const ERR_CANNOT_LISTEN_SOCKET = 12
Global Const ERR_ALREADY_CONNECTED = 13
Global Const ERR_ALREADY_LISTENING = 14
Global Const ERR_LISTENING_REMOTE_PORT = 15
Global Const ERR_LISTENING_HOST_ADDR = 16
Global Const ERR_NOT_LISTENING = 17
Global Const ERR_CANNOT_ACCEPT = 18
Global Const ERR_CONNECT_TO_RECV = 19
Global Const ERR_CONNECT_TO_RECV_COUNT = 20
Global Const ERR_CONNECT_TO_SEND = 21
Global Const ERR_CANNOT_SEND = 22
Global Const ERR_NO_OPTION_VALUE = 23
Global Const ERR_CANNOT_GET_LOCAL = 24
Global Const ERR_UNABLE_TO_LOAD = 25
Global Const ERR_CANNOT_RECV_OOB = 26
Global Const ERR_CANNOT_SEND_OOB = 27
Global Const ERR_INVALID_PROTOCOL = 28
Global Const ERR_FW_SERVER_NOT_DEFINED = 29
Global Const ERR_FW_PORT_NOT_DEFINED = 30
Global Const ERR_REMOTE_PORT_NOT_DEFINED = 31
Global Const ERR_HOST_NOT_DEFINED = 32
Global Const ERR_CANNOT_CONNECT = 33
Global Const ERR_CHANGE_FW_PORT = 34
```

```
' Windows Sockets error codes
Global Const WSABASEERR = 10000
Global Const WSAEINTR = (WSABASEERR+4)
Global Const WSAEBADF = (WSABASEERR+9)
Global Const WSAEACCES = (WSABASEERR+13)
Global Const WSAEFAULT = (WSABASEERR+14)
Global Const WSAEINVAL = (WSABASEERR+22)
Global Const WSAEMFILE = (WSABASEERR+24)
Global Const WSAEWOULDBLOCK = (WSABASEERR+35)
Global Const WSAEINPROGRESS = (WSABASEERR+36)
Global Const WSAEALREADY = (WSABASEERR+37)
Global Const WSAENOTSOCK = (WSABASEERR+38)
Global Const WSAEDESTADDRREQ = (WSABASEERR+39)
Global Const WSAEMSGSIZE = (WSABASEERR+40)
Global Const WSAEPROTOTYPE = (WSABASEERR+41)
Global Const WSAENOPROTOOPT = (WSABASEERR+42)
Global Const WSAEPROTONOSUPPORT = (WSABASEERR+43)
Global Const WSAESOCKTNOSUPPORT = (WSABASEERR+44)
Global Const WSAEOPNOTSUPP = (WSABASEERR+45)
Global Const WSAEPFNOSUPPORT = (WSABASEERR+46)
Global Const WSAEAFNOSUPPORT = (WSABASEERR+47)
Global Const WSAEADDRINUSE = (WSABASEERR+48)
Global Const WSAEADDRNOTAVAIL = (WSABASEERR+49)
Global Const WSAENETDOWN = (WSABASEERR+50)
Global Const WSAENETUNREACH = (WSABASEERR+51)
Global Const WSAENETRESET = (WSABASEERR+52)
Global Const WSAECONNABORTED = (WSABASEERR+53)
Global Const WSAECONNRESET = (WSABASEERR+54)
Global Const WSAENOBUFS = (WSABASEERR+55)
Global Const WSAEISCONN = (WSABASEERR+56)
Global Const WSAENOTCONN = (WSABASEERR+57)
Global Const WSAESHUTDOWN = (WSABASEERR+58)
Global Const WSAETOOMANYREFS = (WSABASEERR+59)
Global Const WSAETIMEDOUT = (WSABASEERR+60)
Global Const WSAECONNREFUSED = (WSABASEERR+61)
Global Const WSAELOOP = (WSABASEERR+62)
Global Const WSAENAMETOOLONG = (WSABASEERR+63)
Global Const WSAEHOSTDOWN = (WSABASEERR+64)
Global Const WSAEHOSTUNREACH = (WSABASEERR+65)
Global Const WSAENOTEMPTY = (WSABASEERR+66)
Global Const WSAEPROCLIM = (WSABASEERR+67)
Global Const WSAEUSERS = (WSABASEERR+68)
Global Const WSAEDQUOT = (WSABASEERR+69)
Global Const WSAESTALE = (WSABASEERR+70)
Global Const WSAEREMOTE = (WSABASEERR+71)
Global Const WSASYSNOTREADY = (WSABASEERR+91)
Global Const WSAVERNOTSUPPORTED = (WSABASEERR+92)
Global Const WSANOTINITIALISED = (WSABASEERR+93)
Global Const WSAHOST_NOT_FOUND = (WSABASEERR+1001)
Global Const WSATRY_AGAIN = (WSABASEERR+1002)
Global Const WSANO_RECOVERY = (WSABASEERR+1003)
Global Const WSANO_DATA = (WSABASEERR+1004)
```



## 2 Windows Sockets Properties

### 2.1 Action

#### Summary

Connect, listen, accept or disconnect on a socket.

#### Description

The Action property controls the connection state of the Windows Socket ActiveX control. A session can be established (directly or via a firewall), accepted or closed by assigning one of the following values to the property.

Value	Action
ACTION_CONNECT	Establish connection.
ACTION_LISTEN	Listen for incoming connection request.
ACTION_ACCEPT	Accept an incoming connection request.
ACTION_DISCONNECT	Close connection.
ACTION_FW_CONNECT	Establish connection via a firewall.

This property can be changed at run time only. There is no default value for this property.

Before setting the Action property to ACTION\_CONNECT or ACTION\_LISTEN, the following properties must be initialized. The Protocol property must be set to select the TCP or UDP transport protocol. The HostAddress property must be set (for ACTION\_CONNECT only) to the internet address of the remote machine. The HostConvert property can be used to convert a host name (or a host address in dotted decimal notation) specified by the HostName property into an internet address.

For a TCP or UDP socket, the LocalPort property must be set to the local port to be associated with the socket. Setting the LocalPort property to 0 will cause the control to use a default local port. Default ports cannot be used for server TCP sockets (when setting the Action property to ACTION\_LISTEN). For a client TCP socket, the RemotePort property must be set to the port on which the remote service to connect to is running. For a server TCP socket, the remote port is not known until a connection request from a remote machine has been accepted. For a UDP socket, the RemotePort property is not used while connecting and can be changed at any time after setting the Action property to ACTION\_CONNECT.

If the local machine is located on a different subnet than the remote machine and the only form of communication between these two machines is through a firewall gateway, then the built-in firewall support of the Windows Socket ActiveX control can be used to establish a session. To establish a connection with a remote host through a firewall, set the Action property to ACTION\_FW\_CONNECT. Before setting the Action property to ACTION\_FW\_CONNECT, some must be initialized. The FirewallServer property must be set to the name or internet address (in dotted decimal notation) of the firewall server. The FirewallPort property must be set to the firewall service port. In addition to the FirewallServer and FirewallPort properties, the HostName and RemotePort properties must also be set as mentioned above. The Protocol property must be set to the TCP transport protocol. Connections via a firewall cannot be made using the UDP transport protocol. Depending on the type of address specified in the HostName property the FwAddrType property must be accordingly set, if the HostName property contains the IP address of the remote host then the FwAddrType must be set to FW\_ADDR\_IP4 and if contains a machine name then the FwAddrType property must contain FW\_ADDR\_DNS. The Distinct Windows Socket ActiveX Control supports both SOCKS version 5 and SOCKS version 4, the application can specify the SOCKS version in the FwSocksVer property. If the SOCKS version is 5 then the application can

specify a authentication method in the FwAuthMethods property, currently only the Username/Password (FwUsername and FwPassword) authentication protocol is supported.

If the connection can be established with the ACTION\_CONNECT or ACTION\_FW\_CONNECT operation, then the OnConnect event will be fired. If the connection cannot be established, then the OnError event will be fired. These events will occur before the next statement (i.e. the statement following the assignment of ACTION\_CONNECT or ACTION\_FW\_CONNECT to the Action property) is executed.

If the application wishes to establish connection using an asynchronous connect operation, the Option property must be set to OPTION\_ASYNC\_CONNECT\_ON before setting the Action property to ACTION\_CONNECT. The ACTION\_CONNECT does not wait until connection is established but returns immediately after issuing a connection request. If the connection can be established, the OnConnect event will be fired; otherwise, an OnError event will occur. Check the Error property to determine the error code returned by the underlying protocol stack.

The application should set a flag in the OnConnect and OnError events, so that it can determine if the session has been established or not. In addition, the application may want to display an error message in the OnError event to inform the user that the connection has not been established. Please check the reference page of the OnError event for a complete listing of error codes.

The Action property can only be set to ACTION\_ACCEPT if an OnAccept event has occurred for a server TCP socket. A server TCP socket is created by setting the Action property to ACTION\_LISTEN. No event is generated in response to a successful ACTION\_LISTEN action.

Once a connection is no longer needed, the session can be terminated by setting the Action property to ACTION\_DISCONNECT. This operation needs to be used to close the connection whether it was established with ACTION\_CONNECT or with ACTION\_FW\_CONNECT. An application must close all connections it has created before it quits.

The Connect, FwConnect, Listen, Accept and Disconnect methods accomplish the same as the above actions. Please check the reference pages of these methods for more detailed information on their usage.

### Example

```
Socket.HostName = "snowy"
Socket.RemotePort = 13
Socket.HostConvert = CONVERT_NAME_TO_ADDRESS
Socket.Action = ACTION_CONNECT
```

## 2.2 BinaryData

### Summary

Contains the binary data received following a call to the ReceiveB method.

### Description

After each call to the ReceiveB method in a C# application this property needs to be read to access binary data.

### Example

```
Private void Skt_OnReceive (object sender, System.EventArgs e)
{
    int len;
    int bytes;
    Object arrData = new byte[bytes];

    Len = axSkt1.ReceiveB (arrData);
```

```
If (len > 0)
{
    object pBuf = new Byte[len];
    pBuf = axSkt1.BinaryData;
    byte []RcvByte = (byte [])pBuf;
    .....
    .....
}
}
```

## 2.3 Convert

### Summary

Convert protocol name to number and vice versa or service name to service port and vice versa.

### Description

The Convert property is used to convert a protocol name to a number and vice versa or a service name to a port and vice versa. The Convert property can be set to one of the following values.

Value	Conversion
PROTO_NAME_TO_NUMBER	ProtocolName to ProtocolNumber.
PROTO_NUMBER_TO_NAME	ProtocolNumber to ProtocolName.
SVC_NAME_TO_PORT	ServiceName to ServicePort.
SVC_PORT_TO_NAME	ServicePort to ServiceName.

If Convert is set to PROTO\_NAME\_TO\_NUMBER, then the control will convert the protocol name given by the ProtocolName property into the protocol number associated with the protocol name. The result of this lookup will be stored in the ProtocolNumber property. If the given protocol name cannot be resolved, then the ProtocolNumber property will be set to 0.

If Convert is set to PROTO\_NUMBER\_TO\_NAME, then the control will convert the protocol number given by the ProtocolNumber property into the corresponding protocol name. The result of this lookup will be stored in the ProtocolName property. If the given protocol number cannot be resolved, then the ProtocolName property will be cleared.

If the Convert property is set to either SVC\_NAME\_TO\_PORT or SVC\_PORT\_TO\_NAME, then the name of the protocol to use when contacting the service must also be specified in the ProtocolName property.

If Convert is set to SVC\_NAME\_TO\_PORT, then the control will convert the service name given by the ServiceName property (associated with the protocol specified by the ProtocolName property) into the service port where the service can be contacted. The result of this lookup will be stored in the ServicePort property. If the given service name cannot be resolved, then the ServicePort property will be set to 0.

If Convert is set to SVC\_PORT\_TO\_NAME, then the control will convert the service port given by the ServicePort property (associated with the protocol specified by the ProtocolName property) into the corresponding service name. The result of this lookup will be stored in the ServiceName property. If the given service port cannot be resolved, then the ServiceName property will be cleared.

The Error property can be checked to obtain the error code returned by the underlying protocol stack.

The ProtoNameToNumber, ProtoNumberToName, SvcNametoPort and SvcPortToName methods accomplish the same as the above actions. Please check the reference pages of these methods for more detailed information on their usage.

This property can be changed at run time only. There is no default value for this property.

### Example

```
Socket.ProtocolName = "tcp"
Socket.Convert = PROTO_NAME_TO_NUMBER
' Socket.ProtocolNumber will now contain the protocol number for "tcp".
```

```
Socket.ServiceName = "daytime"
```

```
Socket.ProtocolName = "tcp"
```

```
Socket.Convert = SVC_NAME_TO_PORT
```

```
' Socket.ServicePort will now contain the port number of the "daytime" service for "tcp" protocol.
```

## 2.4 Error

### Summary

Error code returned by underlying protocol stack.

### Description

The Error property specifies the error code returned by the protocol stack if a socket operation could not be completed successfully. This property is read only.

The Error property reflects the error code of the last socket operation initiated by setting the Action, Option, Send, Convert or HostConvert properties or by accessing the Receive and ReceiveCount properties.

The value of this property must be checked immediately after initiating a socket operation. Accessing other properties may change the value of this property.

This property can be read at any time. There is no default value for this property.

The following is a list of possible error codes returned by the Windows Sockets protocol stack. Some of the error codes are not very common and hence, are not listed here. Please check the documentation for Windows Sockets for further information on these error codes.

<b>Constant</b>	<b>Value</b>	<b>Meaning</b>
WSAEINTR	10004	Interrupted function call.
WSAEACCES	10013	Permission denied.
WSAEFAULT	10014	Bad address.
WSAEINVAL	10022	Invalid argument.
WSAEMFILE	10024	Too many open files
WSAEWOULDBLOCK	10035	Resource temporarily unavailable.
WSAEINPROGRESS	10036	Operation now in progress.
WSAEALREADY	10037	Operation already in progress.
WSAENOTSOCK	10038	Socket operation on non-socket.
WSAEDESTADDRREQ	10039	Destination address required.
WSAEMSGSIZE	10040	Message too long.
WSAEPROTOTYPE	10041	Protocol wrong type for socket.
WSAENOPROTOOPT	10042	Bad protocol option.
WSAEPROTONOSUPPORT	10043	Protocol not supported.
WSAESOCKTNOSUPPORT	10044	Socket type not supported.
WSAEOPNOTSUPP	10045	Operation not supported.
WSAEPFNOSUPPORT	10046	Protocol family not supported.
WSAEAFNOSUPPORT	10047	Address family not supported by protocol family.
WSAEADDRINUSE	10048	Address already in use.
WSAEADDRNOTAVAIL	10049	Cannot assign requested address.
WSAENETDOWN	10050	Network is down.
WSAENETUNREACH	10051	Network is unreachable.
WSAENETRESET	10052	Network dropped connection on reset.
WSAECONNABORTED	10053	Software caused connection abort.
WSAECONNRESET	10054	Connection reset by peer.
WSAENOBUFS	10055	No buffer space available.
WSAEISCONN	10056	Socket is already connected.
WSAENOTCONN	10057	Socket is not connected.
WSAESHUTDOWN	10058	Cannot send after socket shutdown.
WSAETIMEDOUT	10060	Connection timed out.
WSAECONNREFUSED	10061	Connection refused.
WSAEHOSTDOWN	10064	Host is down.

WSAEHOSTUNREACH	10065	No route to host.
WSAEPROCLIM	10066	Too many processes.
WSASYSNOTREADY	10091	Network subsystem is unavailable.
WSAVERNOTSUPPORTED	10092	WINSOCK.DLL version out of range.
WSANOTINITIALISED	10093	Successful WSAShutdown() not yet performed.
WSAHOST_NOT_FOUND	11001	Host not found.
WSATRY_AGAIN	11002	Non-authoritative host not found.
WSANO_RECOVERY	11003	This is a non-recoverable error.
WSANO_DATA	11004	Valid name, no data record of requested type.

The following describes each error in more detail.

#### **WSAEINTR**

A blocking operation was interrupted by a call to WSACancelBlockingCall().

#### **WSAEACCES**

An attempt was made to access a socket in a way forbidden by its access permissions. An example is using a broadcast address for sendto() without broadcast permission being set using setsockopt(SO\_BROADCAST)

#### **WSAEFAULT**

The system detected an invalid pointer address in attempting to use a pointer argument of a call. This occurs if an application passes an invalid pointer value, or if the length of the buffer is too small. For instance, if the length of a struct sockaddr is smaller than sizeof(struct sockaddr).

#### **WSAEINVAL**

Some invalid argument was supplied (for example, specifying an invalid level to the setsockopt() function). In some instances, it also refers to the current state of the socket - for instance, calling accept() on a socket that is not listen()ing.

#### **WSAEMFILE**

Too many open sockets. Each implementation may have a maximum number of socket handles available, either globally, per process or per thread.

#### **WSAEWOULDBLOCK**

This error is returned from operations on non-blocking sockets that cannot be completed immediately, for example recv() when no data is queued to be read from the socket. It is a non-fatal error, and the operation should be retried later. It is normal for WSAEWOULDBLOCK to be reported as the result from calling connect() on a non-blocking SOCK\_STREAM socket, since some time must elapse for the connection to be established.

#### **WSAEINPROGRESS**

A blocking operation is currently executing. Windows Sockets only allows a single blocking operation to be outstanding per task (or thread), and if you make any other function call (whether or not it references that or any other socket) the function fails with the WSAEINPROGRESS error.

#### **WSAEALREADY**

An operation was attempted on a non-blocking socket that already had an operation in progress - i.e.

calling `connect()` a second time on a non-blocking socket that is already connecting, or canceling an asynchronous request (`WSAAsyncGetXbyY()`) that has already been canceled or completed.

**WSAENOTSOCK**

An operation was attempted on something that is not a socket. Either the socket handle parameter did not reference a valid socket, or for `select()`, a member of an `fd_set` was not valid.

**WSAEDESTADDRREQ**

A required address was omitted from an operation on a socket. For example, if `sendto()` is called with the remote address as `INADDR_ANY`.

**WSAEMSGSIZE**

A message sent on a datagram socket was larger than the internal message buffer or some other network limit, or the buffer used to receive a datagram into was smaller than the datagram itself.

**WSAEPROTOTYPE**

A protocol was specified in the `socket()` function call that does not support the semantics of the socket type requested. For example, you cannot specify the ARPA Internet UDP protocol with a socket type of `SOCK_STREAM`.

**WSAENOPROTOPT**

An unknown, invalid or unsupported option or level was specified in a `getsockopt()` or `setsockopt()` call.

**WSAEPROTONOSUPPORT**

The requested protocol has not been configured into the system, or no implementation for it exists. For example, a `socket()` call requests a `SOCK_DGRAM` socket, but specifies a stream protocol.

**WSAESOCKTNOSUPPORT**

The support for the specified socket type does not exist in this address family. For example, the optional type `SOCK_RAW` might be selected in a `socket()` call, and the implementation does not support `SOCK_RAW` sockets at all.

**WSAEOPNOTSUPP**

The attempted operation is not supported for the type of object referenced. Usually, this occurs when a socket descriptor to a socket that cannot support this operation, for example, trying to accept a connection on a datagram socket.

**WSAEPFNOSUPPORT**

The protocol family has not been configured into the system or no implementation for it exists. Has a slightly different meaning to `WSAEAFNOSUPPORT`, but is interchangeable in most cases, and all Windows Sockets functions that return one of these specify `WSAEAFNOSUPPORT`.

**WSAEAFNOSUPPORT**

An address incompatible with the requested protocol was used. All sockets are created with an associated "address family" (i.e. `AF_INET` for Internet Protocols) and a generic protocol type (i.e. `SOCK_STREAM`). This error will be returned if an incorrect protocol is explicitly requested in the `socket()` call, or if an address of the wrong family is used for a socket, e.g. in `sendto()`.

**WSAEADDRINUSE**

Only one usage of each socket address (protocol/IP address/port) is normally permitted. This occurs if an application attempts to `bind()` a socket to an IP address/port that has already been used for an existing socket, or a socket that wasn't closed properly, or one that is still in the process of closing. For server applications that need to `bind()` multiple sockets to the same port number, consider using `setsockopt(SO_REUSEADDR)`. Client applications usually need not call `bind()` at all - `connect()` will choose an unused port automatically.

**WSAEADDRNOTAVAIL**

The requested address is not valid in its context. Normally, results from an attempt to bind() to an address that is not valid for the local machine, or connect()/sendto() an address or port that is not valid for a remote machine (e.g. port 0).

**WSAENETDOWN**

A socket operation encountered a dead network. This could indicate a serious failure of your network system (i.e. the protocol stack that the WINSOCK DLL runs over), the network interface, or the local network itself.

**WSAENETUNREACH**

A socket operation was attempted to an unreachable network. This usually means the local software knows no route to reach the remote host.

**WSAENETRESET**

The host you were connected to crashed and rebooted. May also be returned by setsockopt() if you try to set SO\_KEEPAALIVE on a connection that has already failed.

**WSAECONNABORTED**

An established connection was aborted by the software in your host machine, possibly due to a data transmission timeout or protocol error.

**WSAECONNRESET**

A existing connection was forcibly closed by the remote host. This normally results if the peer application on the remote host is suddenly stopped, the host is rebooted, or the remote host used a "hard close" (see setsockopt(SO\_LINGER)) on the remote socket.

**WSAENOBUFS**

An operation on a socket could not be performed because the system lacked sufficient buffer space or because a queue was full.

**WSAEISCONN**

A connect request was made on an already connected socket. Some implementations also return this error if sendto() is called on a connected SOCK\_DGRAM socket (For SOCK\_STREAM sockets, the to parameter in sendto() is ignored), although other implementations treat this as a legal occurrence.

**WSAENOTCONN**

A request to send or receive data was disallowed because the socket is not connected and (when sending on a datagram socket using sendto()) no address was supplied. Any other type of operation might also return this error - for example, setsockopt() setting SO\_KEEPAALIVE if the connection has been reset.

**WSAESHUTDOWN**

A request to send or receive data was disallowed because the socket had already been shut down in that direction with a previous shutdown() call. By calling shutdown() you do a partial close of a socket, which means you have discontinued sending or receiving or both.

**WSAETIMEDOUT**

A connection attempt failed because the connected party did not properly respond after a period of time.

**WSAECONNREFUSED**

No connection could be made because the target machine actively refused it. This usually results from trying to connect to a service that is inactive on the foreign host - there is no server application running.

**WSAEHOSTDOWN**

A socket operation failed because the destination host was down. A socket operation encountered a dead host. Networking activity on the local host has not been initiated. These conditions are more likely to be indicated by the error WSAETIMEDOUT.

**WSAEHOSTUNREACH**

A socket operation was attempted to an unreachable host. See WSAENETUNREACH

**WSAEPROCLIM**

A Windows Sockets implementation may have a limit on the number of applications that may use it simultaneously. WSASStartup() may fail with this error if the limit has been reached.

**WSASYSNOTREADY**

This error is returned by WSASStartup() if the Windows Sockets implementation cannot function at this time, because the underlying system it uses to provide network services is currently unavailable.

Users should check:

- \* that the WINSOCK.DLL file is in the current path,
- \* that the WINSOCK.DLL file is from the same vendor as your underlying protocol stack. You cannot mix and match (WINSOCK DLLs must be supplied by the same vendor that provided your underlying protocol stack).
- \* that you are not trying to use more than one WinSock implementation simultaneously. If you have more than one WINSOCK DLL on your system, be sure the first one in the path is appropriate for the network subsystem currently loaded.
- \* your WinSock implementation documentation to be sure all necessary components are currently installed and configured correctly.

**WSAVERNOTSUPPORTED**

The current WinSock implementation does not support the Windows Sockets specification version requested by the application. Check that no old WINSOCK.DLL files are being accessed, or contact your vendor to see if an updated WINSOCK.DLL exists.

**WSANOTINITIALISED**

Either your application hasn't called WSASStartup(), or WSASStartup() failed. You may be accessing a socket which the current active task does not own (i.e. you're trying to share a socket between tasks), or you may have already called WSACleanup() too many times.

**WSAHOST\_NOT\_FOUND**

No such host is known. The name you have used is not an official hostname or alias, or it cannot be found in the database(s) being queried. This error may also be returned for protocol and service queries, and means the specified name could not be found in the relevant database.

**WSATRY\_AGAIN**

This is usually a temporary error during hostname resolution and means that the local server did not receive a response from an authoritative server. A retry at some time later may be successful.

**WSANO\_RECOVERY**

This indicates some sort of non-recoverable error occurred during a database lookup. This may be because the database files (e.g. BSD-compatible HOSTS, SERVICES or PROTOCOLS files) could not be found, or a DNS request was returned by the server with a severe error.

**WSANO\_DATA**

The requested name is valid and was found in the database, but it does not have the correct associated data being resolved for. The usual example for this is a hostname -> address translation attempt (using gethostbyname() or WSAAsyncGetHostByName()) which uses the DNS (Domain Name Server), and an MX record is returned but no A record - indicating the host itself exists, but is not directly reachable.

**Example**

```
Const WSAENOTCONN = 10057
```

```
Socket.Send = "test"
```

```
If Socket.Error = WSAENOTCONN Then
```

```
    MsgBox "Connection closed", 64, "Sample Program"
```

```
End If
```

## 2.5 FirewallPort

### Summary

Firewall server port.

### Description

The FirewallPort property specifies the port on the firewall server through which a connection should be established. This property is used for client TCP sockets. Client TCP sockets must set the FirewallPort property just once before a connection is established. Once connected, the firewall port remains the same for the entire session.

The FirewallPort property must be set before a connection is established with a client TCP socket (by setting the Action property to ACTION\_FW\_CONNECT). The Windows Socket ActiveX control does not verify the setting of the FirewallPort property and any value (except 0) is therefore legal.

This property can be changed at any time except after a connection has been established with a client TCP socket (by setting the Action property to ACTION\_CONNECT or ACTION\_FW\_CONNECT). The default value for this property is 1080.

### Example

```
Socket.FirewallServer = "127.43.101.10"  
Socket.FirewallPort = 1080  
Socket.HostName = "127.43.101.12"  
Socket.RemotePort = 13  
Socket.Action = ACTION_FW_CONNECT
```

## 2.6 FirewallServer

### Summary

Name of firewall server or dotted decimal internet address.

### Description

The FirewallServer property specifies the name or internet address of a firewall through which a connection is to be made. This property must be set before a connection can be established (by changing the Action property). There are three possible ways of specifying a firewall server name.

#### Machine Name

An application only needs to specify the name of the firewall server if the host is located on the same network as the local PC or if the internet address of the host is defined in the local hosts data base. If the firewall server is not on the local network, then the underlying protocol will route the traffic through a gateway. If the host is not defined in the local hosts data base, then the underlying protocol will contact the name server to resolve the internet address of the firewall server.

#### Machine and Domain Name

An application needs to specify the machine name and the domain name if the host is not located on the same network as the local PC. Fully domain qualified machine names are written from left to right in ascending order (for example, *speedy.distinct.com*). If both machine and domain names are specified, then the underlying protocol will contact the name server to resolve the internet address of the firewall server.

#### Internet Address

Sometimes the user knows only the internet address of the firewall server that he or she wants to use. In this case, the internet address can be entered in what is known as the dotted decimal notation (for example, *127.43.101.12*). If the host identified by this address is not on the local network, then the underlying protocol will route the traffic through a gateway.

This property can be changed at design time and at run time before a connection has been established. There is no default value for this property.

### Example

```
Socket.FirewallServer = "127.43.101.10"  
Socket.FirewallPort = 1080  
Socket.HostName = "127.43.101.12"  
Socket.RemotePort = 13  
Socket.Action = ACTION_FW_CONNECT
```

## 2.7 FwAddrType

### Summary

Address type of the destination host.

### Description

The *FwAddrType* property specifies the address type in the *HostName* property. This property must be set before a connection can be established (by changing the Action property to ACTION\_FW\_CONNECT or calling the method FwConnect). There are three possible ways of specifying a destination address in the *HostName* Property and therefore the *FwAddrType* property can have three possible values:

FW_ADDR_IP4	Address is a version 4 IP address
FW_ADDR_DNS	Address is a DNS style domain name
FW_ADDR_IP6	Address is a version 6 IP address

The *FwAddrType* property must be set to FW\_ADDR\_IP4 if the application is specifying a Internet address in the dotted decimal notation (*198.201.122.11*) in the *HostName* property. If the application wants to specify the machine name or machine name and domain name (for example *speedy.distinct.com*) as the *HostName* they must set the *FwAddrType* property to FW\_ADDR\_DNS.

Note that the FW\_ADDR\_IP6 is not supported currently.

This property can be changed at design time and at run time before a connection has been established.

The default value for this property is FW\_ADDR\_IP4.

### Example

```
Socket.FirewallServer = "sparky.distinct.com"  
Socket.FirewallPort = 1080  
Socket.FwAddrType = FW_ADDR_DNS  
Socket.HostName = "speedy.distinct.com"  
Socket.Action = ACTION_FW_CONNECT
```

## 2.8 FwAuthMethods

### Summary

Firewall authentication methods.

### Description

The *FwAuthMethods* property specifies the authentication method that can be used when connecting to the firewall server. This property must be set before a connection can be established (by changing the Action property to ACTION\_FW\_CONNECT or calling the method FwConnect). The *FwAuthMethods* can be "0", "1" or "2" or a combination of "0", "1", "2", for example "01" or "12". "0" means that no authentication is required, "1" means GSSAPI and "2" means that a valid username and password is required. Currently only methods "0" and "2" are supported.

This property can be changed at design time and at run time before a connection has been established.

The default value for this property is "0".

### Example

```
Socket.FirewallServer = "sparky.distinct.com"  
Socket.FirewallPort = 1080  
Socket.FwAddrType = FW_ADDR_DNS  
Socket.HostName = "speedy.distinct.com"  
Socket.FwAuthMethods = "2"  
Socket.FwUsername = "joe"  
Socket.FwPassword = "distinct"  
Socket.Action = ACTION_FW_CONNECT
```

## 2.9 FwPassword

### Summary

A valid password .

### Description

The *FwPassword* property specifies a valid password that is required during authentication. A valid password is required when connecting to SOCKS version 5 server if the authentication method (*FwAuthMethods*) specified is "2" (Username/Password authentication protocol).

This property can be changed at design time and at run time before a connection has been established by setting the *Action* property to ACTION\_FW\_CONNECT or by calling the method FwConnect.

The property does not have any default value.

### Example

```
Socket.FirewallServer = "sparky.distinct.com"  
Socket.FirewallPort = 1080  
Socket.FwAddrType = FW_ADDR_DNS  
Socket.HostName = "speedy.distinct.com"  
Socket.FwAuthMethods = "2"  
Socket.FwUsername = "joe"  
Socket.FwPassword = "distinct"  
Socket.Action = ACTION_FW_CONNECT
```

## 2.10 FwSocksVer

### Summary

The firewall server version.

### Description

The *FwSocksVer* property is used specify the version of the SOCKS server. This property can have any one or a combination of the following values:

Value	Meaning
FW_VERSION5	The SOCKS version is 5.
FW_VERSION4	The SOCKS version is 4

If the firewall server version is unknown then the application can specify both FW\_VERSION5 and FW\_VERSION4. The Distinct Windows Socket control will automatically detect the firewall server version and make the appropriate connection.

This property can be set at design time or at run time before a connection is established by calling the FwConnect method (or by setting the Action to ACTION\_FW\_CONNECT).

### Example

```
Socket.FirewallServer = "sparky.distinct.com"  
Socket.FirewallPort = 1080  
Socket.FwAddrType = FW_ADDR_DNS  
Socket.HostName = "speedy.distinct.com"  
Socket.FwAuthMethods = "2"  
Socket.FwUsername = "joe"  
Socket.FwPassword = "distinct"  
Socket.FwSocksVer = FW_VERSION5  
Socket.Action = ACTION_FW_CONNECT
```

## 2.11 FwUsername

### Summary

A valid user name.

### Description

The *FwUsername* property specifies a valid username. A valid user id is required when connecting to SOCKS version 5 server if the authentication method (*FwAuthMethods*) specified is "2" (Username/Password authentication protocol). It is mandatory when a connection is established with SOCKS version 4 server.

This property can be changed at design time and at run time before a connection has been established by setting the Action property to ACTION\_FW\_CONNECT (or by calling the FwConnect method).

The property does not have any default value.

### Example

```
Socket.FwSocksVer = FW_VERSION5
Socket.FirewallServer = "sparky.distinct.com"
Socket.FirewallPort = 1080
Socket.FwAddrType = FW_ADDR_DNS
Socket.HostName = "speedy.distinct.com"
Socket.FwAuthMethods = "2"
Socket.FwUsername = "joe"
Socket.FwPassword = "distinct"
Socket.Action = ACTION_FW_CONNECT
```

## 2.12 HostAddress

### Summary

Resolved internet address of host.

### Description

The HostAddress property is used to specify the internet address of the remote machine. This property is generally not modified directly. Instead, the HostConvert property is used to convert a host name (or a host address in dotted decimal notation) specified by the HostName property into an internet address.

The HostAddress property is specified as a binary long address and **not** as a dotted decimal IP address.

To obtain the dotted decimal IP address given a host name, the application must first convert the host name specified by the HostName property into an internet address and then convert the internet address contained in the HostAddress property into an IP address in dotted decimal notation.

Before a connection can be established (by changing the Action property) over any protocol, the HostAddress property must contain a valid internet address instead of the default value of 0.

The HostAddress property can be used independent of the Action property to convert between host addresses and host names for informational purposes. Since resolving a host name or address may involve sending queries over the network to a name server, the HostConvert property can only be used at run time.

### Example

```
Socket.HostName = "speedy.distinct.com"  
Socket.HostConvert = CONVERT_NAME_TO_ADDRESS  
If Socket.HostAddress = 0 Then  
    MsgBox "Unable to resolve host name ", 64, "Sample Program"  
End If
```

## 2.13 HostConvert

### Summary

Convert host name to address or address to host name or address to dotted decimal.

### Description

The HostConvert property is used to convert a host name to a host address and vice versa and can be set to one of the following values.

Value	Conversion
CONVERT_NAME_TO_ADDRESS	HostName to HostAddress.
CONVERT_ADDRESS_TO_NAME	HostAddress to HostName.
CONVERT_ADDRESS_TO_DECIMAL	HostAddress to dotted decimal HostName.

If HostConvert is set to CONVERT\_NAME\_TO\_ADDRESS, then the control will convert the host name (or the host address specified in dotted decimal notation) given with the HostName property into the internet address of the host. The result of this lookup will be stored in the HostAddress property. If the given host name cannot be resolved, then the HostAddress property will be set to 0.

If HostName contains an address in dotted decimal notation (for example, 127.43.101.12), then the control will simply convert the string to an internet address. Otherwise, the control first checks if the specified host is defined in the local hosts data base. If the host is not found, then the control will send queries over the network to a name server.

If HostConvert is set to CONVERT\_ADDRESS\_TO\_NAME, then the control will convert the internet address given with the HostAddress property into the name of the host. The result of this lookup will be stored in the HostName property. If the given host address cannot be resolved, then the HostName property will be cleared.

The control first checks if the specified internet address is defined in the local hosts data base. If the address is not found, then the control will send queries over the network to a name server.

If HostConvert is set to CONVERT\_ADDRESS\_TO\_DECIMAL, then the control will convert the internet address given with the HostAddress property into the dotted decimal address of the host. The result of this lookup will be stored in the HostName property. If the given host address is 0, then the HostName property will be cleared.

The NameToAddr, AddrToName and AddrToDecimal methods accomplish the same as the above actions. Please check the reference pages of these methods for more detailed information on their usage.

This property can be changed at run time only. There is no default value for this property.

### Example

```
Socket.HostName = "speedy"  
Socket.RemotePort = 13  
Socket.HostConvert = CONVERT_NAME_TO_ADDRESS  
Socket.Action = ACTION_CONNECT
```

## 2.14 HostName

### Summary

Name of host or dotted decimal internet address.

### Description

The HostName property specifies the name or internet address of a remote machine. This property must be set and converted into an internet address with the HostConvert property before a connection can be established (by changing the Action property). There are three possible ways of specifying a host name.

#### Machine Name

An application only needs to specify the name of the remote machine if the host is located on the same network as the local PC or if the internet address of the host is defined in the local hosts data base. If the remote machine is not on the local network, then the underlying protocol will route the traffic through a gateway. If the host is not defined in the local hosts data base, then the underlying protocol will contact the name server to resolve the internet address of the remote machine.

#### Machine and Domain Name

An application needs to specify the machine name and the domain name if the host is not located on the same network as the local PC. Fully domain qualified machine names are written from left to right in ascending order (for example, *speedy.distinct.com*). If both machine and domain names are specified, then the underlying protocol will contact the name server to resolve the internet address of the remote machine.

#### Internet Address

Sometimes the user knows only the internet address of the remote machine that he or she wants to use. In this case, the internet address can be entered in what is known as the dotted decimal notation (for example, *127.43.101.12*). If the host identified by this address is not on the local network, then the underlying protocol will route the traffic through a gateway.

This property can be changed at design time and at run time before a connection has been established. The given host name must be converted into an internet address using the HostConvert property before the Action property can be set to ACTION\_CONNECT.

There is no default value for this property.

### Example

```
Socket.HostName = "127.43.101.12"  
Socket.RemotePort = 13  
Socket.HostConvert = CONVERT_NAME_TO_ADDRESS  
Socket.Action = ACTION_CONNECT
```

## 2.15 Local

### Summary

Name or dotted decimal internet address of local host.

### Description

The Local property specifies the name or internet address of the local machine. This property can only be accessed during run time and can not be changed by the application.

Depending on the underlying protocol stack and how it is configured, the Local property may contain a simple host name or a fully qualified domain name.

To obtain the internet address of the local machine, the host name (or the local dotted decimal internet address) in the Local property can be assigned to the HostName property. Then, the HostConvert property can be used to obtain the corresponding internet address.

This is a read-only property. There is no default value for this property.

### Example

```
Dim LocalHost As String
```

```
LocalHost = Socket.Local
```

## 2.16 LocalHostAddress

### Summary

Internet address of local host to a connection.

### Description

The LocalHostAddress property specifies internet address of the local machine to bind a connection. This property can only be set before a connection is established by setting the Action property to ACTION\_CONNECT (or by calling the Connect method) or ACTION\_LISTEN (or by calling the Listen method).

The LocalHostAddress property is used only for a TCP connection.

To obtain the internet address of the local machine, the host name (or the local dotted decimal internet address) in the Local property can be assigned to the HostName property. Then, the HostConvert property can be used to obtain the corresponding internet address.

This property can be set at any time except after a connection has been established. The default value for this property is 0.

### Example

```
Socket.Protocol = PROTOCOL_TCP  
Socket.LocalHostAddress = Address  
Socket.Action = ACTION_LISTEN
```

## 2.17 LocalPort

### Summary

Local port to be used for next connection.

### Description

The LocalPort property is used to specify the local port to which a socket is bound. This port number is the value by which the socket is known to another host. Remote machines can connect and send messages to the local port of a socket.

If the LocalPort property is set to 0, then the control will choose a default port in a predefined range. To avoid problems with stale connections, the default port used is always one higher than the default port used during the last connection. Using a default local port will be sufficient for most applications.

A server TCP socket (created by setting the Action property to ACTION\_LISTEN) cannot use a default local port. Since the remote machine must know to which port it can send a connection request, the local port on which a server TCP socket is listening for requests must be agreed upon by both machines.

This property can be changed at any time before a connection is established (by setting the Action property to ACTION\_CONNECT) or the socket is put into listening mode (by setting the Action property to ACTION\_LISTEN). The default value of this property is 0.

### Example

```
Socket.Protocol = PROTOCOL_TCP  
Socket.LocalPort = 3000  
Socket.Action = ACTION_LISTEN
```

## 2.18 Option

### Summary

Socket option.

### Description

The Option property is used to specify the socket options associated with the Windows Socket ActiveX control. The Option property can be set to one of the following values.

Value	Meaning
OPTION_BROADCAST_ON*	Allow transmission of broadcast messages.
OPTION_BROADCAST_OFF	Do not allow transmission of broadcast messages.
OPTION_LINGER_ON*	Set linger time-out for close socket operation.
OPTION_LINGER_OFF	Do not linger during close socket operation.
OPTION_KEEPA_LIVE_ON	Send keep-alive TCP packets.
OPTION_KEEPA_LIVE_OFF*	Do not send keep-alive TCP packets.
OPTION_REUSEADDR_ON	Allow reuse of local address.
OPTION_REUSEADDR_OFF*	Do not allow reuse of local address.
OPTION_ASYNC_CONNECT_ON	Use asynchronous connect.
OPTION_ASYNC_CONNECT_OFF*	Do not use asynchronous connect.
OPTION_RECV_OOB_ON	Receive out-of-band data separately.
OPTION_RECV_OOB_OFF*	Receive out-of-band data as regular data.

\* Default values

The OPTION\_BROADCAST\_ON and OPTION\_BROADCAST\_OFF options are only available for UDP connections. To allow transmission of broadcast messages on such a socket, the Option property must be set to OPTION\_BROADCAST\_ON. When setting the Option property to OPTION\_BROADCAST\_OFF, broadcast messages can no longer be transmitted on the socket. These two options can be used even after a connection has been established. By default the Windows Socket ActiveX control enables broadcast for UDP connections.

To establish a connection in asynchronous mode, the Option property must be set to OPTION\_ASYNC\_CONNECT\_ON before connecting using ACTION\_CONNECT. This setting will cause ACTION\_CONNECT to return immediately without waiting for the connection to be established. Later, once the connection is established, an OnConnect event will be fired. To turn off asynchronous connect, the Option property must be set to OPTION\_ASYNC\_CONNECT\_OFF before establishing a connection using ACTION\_CONNECT. Connections are not established in asynchronous mode by default.

A socket may by default not be bound to a local address which is already in use. If, however, the application desires to reuse an address, it must set the Option property to OPTION\_REUSEADDR\_ON before trying to establish a connection. Since every connection is uniquely identified by the combination of local and remote ports and addresses, two sockets can be bound to the same local port and address as long as the remote port and address are different. To disable the reuse of addresses, set the Option property to OPTION\_REUSEADDR\_OFF before trying to establish a connection (this is the default setting).

An application may request the underlying protocol stack to enable the use of keep-alive packets on TCP connections (provided the protocol stack supports the use of keep-alives) by setting the Option property to `OPTION_KEEPALIVE_ON`. The send and time-out intervals must be specified as required by the protocol stack in use. To disable keep-alive probes, set the Option property to `OPTION_KEEPALIVE_OFF` (this is the default setting).

The `OPTION_LINGER_ON` and `OPTION_LINGER_OFF` options control the behavior of the socket while disconnecting. If the Option property is set to `OPTION_LINGER_ON`, then the socket is not closed until all data buffered in the send buffer is transmitted or the given time-out has been exceeded. The time-out value in seconds must be specified with the `OptionValue` property. If `OptionValue` is set to 0 with linger enabled, then the connection will be reset. If the Option property is set to `OPTION_LINGER_OFF`, then the underlying protocol stack will close without waiting for buffered data to be sent. The Windows Socket ActiveX control by default enables linger with a three second time-out.

Out-of-band data (or TCP urgent data) can be delivered to the user either separately from regular data or as part of regular data. If an application needs to process out-of-band data apart from regular data, then the Option property must be set to `OPTION_RECV_OOB_ON`. In this case, the arrival of out-of-band data will trigger the `OnReceiveOOB` event. The `ReceiveOOB` property can then be used to access the out-of-band data. By default the `OPTION_RECV_OOB_OFF` setting is enabled and out-of-band data is received as part of the regular data stream. The out-of-band options are available only for TCP connections.

The `Broadcast`, `Linger`, `KeepAlive`, `ReuseAddr`, `AsyncConnect` and `RecvOOB` methods accomplish the same as the above actions. Please check the reference pages of these methods for more detailed information on their usage.

The Option property can be changed at any time before a connection is established. Once connected, only the broadcast, linger and keep alive options can be changed. There is no default value for this property.

### Example

```
Socket.Protocol = PROTOCOL_TCP  
Socket.Option = OPTION_KEEPALIVE_ON
```

## 2.19 OptionValue

### Summary

Socket option value.

### Description

The OptionValue property is used to specify the value for options set using the Option property. Most options do not require a value but are simply enabled or disabled. However, to turn linger on, an application must specify the linger time-out in seconds in the OptionValue property.

Before setting the Option property to OPTION\_LINGER\_ON, the OptionValue property must be set to the value of the desired time-out in seconds. This will direct the underlying protocol stack how long to wait for data in the send buffer to be sent before closing the connection during ACTION\_DISCONNECT. If the OptionValue property is set to 0 with linger enabled, then the connection will be reset during ACTION\_DISCONNECT. The Windows Socket ActiveX control by default enables linger with a three second time-out.

This property can be changed at any time before a connection is established (by setting the Action property to ACTION\_CONNECT) or the socket is put into listening mode (by setting the Action property to ACTION\_LISTEN) and before the connection is closed (by setting the Action property to ACTION\_DISCONNECT). There is no default value of this property.

### Example

```
Socket.OptionValue = 0
Socket.Option = OPTION_LINGER_ON
Socket.Action = ACTION_DISCONNECT           ' connection reset
```

## 2.20 Protocol

### Summary

Protocol to be used for next connection.

### Description

The Protocol property is used to specify which IP protocol a Socket control will be using and can be set to one of the following values.

Value	Protocol
PROTOCOL_TCP	Transmission Control Protocol (TCP, stream).
PROTOCOL_UDP	User Datagram Protocol (UDP, datagram).

The TCP protocol can be used either as a client or as a server. A client TCP socket actively connects from some local port (which may be a default port) to a service on a known remote port by setting the Action property to ACTION\_CONNECT. A server TCP socket is put into listening mode by setting the Action property to ACTION\_LISTEN. When a connection request arrives from another host and the application chooses to accept the connection, then the listening server socket is connected just like a client socket. Once connected, a server socket cannot accept any more connection requests. At the end of a session, both client and server TCP sockets close their connection by setting the Action property to ACTION\_DISCONNECT.

A TCP Socket control can actively connect as a client socket even if it was used as a server socket previously and it can listen for incoming connection requests as a server socket even if it was used as a client socket previously. As long as the connection has been disconnected by setting the Action property to ACTION\_DISCONNECT, the Socket control can perform either function independent from its previous function. A TCP socket cannot, however, be both a client and a server socket at the same time.

The UDP protocol is designed as a peer to peer protocol without a client versus server distinction. A UDP socket is not connected to any particular remote port on another host, but can instead send and receive messages to and from any number of hosts. Each UDP socket is, however, associated with a local port. This unique port number is used to identify different UDP Socket controls on the same machine to other hosts.

This property can be changed at any time before a connection is established (by setting the Action property to ACTION\_CONNECT) or the socket is put into listening mode (by setting the Action property to ACTION\_LISTEN). The default value of this property is PROTOCOL\_TCP.

### Example

```
Socket.HostName = "127.43.101.12"  
Socket.RemotePort = 13  
Socket.Protocol = PROTOCOL_TCP  
Socket.Action = ACTION_CONNECT
```

## 2.21 ProtocolName

### Summary

Official protocol name.

### Description

The ProtocolName property specifies the official name of a protocol. This property is used to look up the corresponding protocol number or to look up a service name or number over the given protocol. This property must be set before setting the Convert property to PROTO\_NAME\_TO\_NUMBER, SVC\_NAME\_TO\_PORT or SVC\_PORT\_TO\_NAME.

The Windows Socket ActiveX control can look up either the protocol name given the number or the protocol number given the name. To convert the protocol name to the number, set the ProtocolName property to the name of the protocol and then set the Convert property to PROTO\_NAME\_TO\_NUMBER. To retrieve the protocol name, set the ProtocolNumber property to the number of the protocol and then set the Convert property to PROTO\_NUMBER\_TO\_NAME.

This property can only be changed at run time. There is no default value for this property.

### Example

```
Socket.ProtocolName = "tcp"  
Socket.Convert = PROTO_NAME_TO_NUMBER  
' Socket.ProtocolNumber will now contain the protocol number for "tcp".
```

## 2.22 ProtocolNumber

### Summary

Official protocol number.

### Description

The ProtocolNumber property specifies the number of the protocol to be used while contacting a service. The protocol number and the protocol name are used to identify the protocol. This property must be set before setting the Convert property to PROTO\_NUMBER\_TO\_NAME.

The Windows Socket ActiveX control can look up either the protocol name given the number or the protocol number given the name. To convert the protocol name to the number, set the ProtocolName property to the name of the protocol and then set the Convert property to PROTO\_NAME\_TO\_NUMBER. To retrieve the protocol name, set the ProtocolNumber property to the number of the protocol and then set the Convert property to PROTO\_NUMBER\_TO\_NAME.

This property can only be changed at run time. There is no default value for this property.

### Example

```
Socket.ProtocolNumber = 0  
Socket.Convert = PROTO_NUMBER_TO_NAME  
'Socket.ProtocolName will now contain "tcp".'
```

## 2.23 Receive

### Summary

Receive buffer.

### Description

The Receive property is used to access data that has been sent by the remote machine. This property can only be accessed by assigning its value to another string while a connection is established. Whenever the property is assigned to a string, as many bytes of data as possible up to the number specified by the ReceiveLen property are copied from the receive buffer into that string.

Usually, this property is read during an OnReceive event. This event informs the application that more data has arrived from the connected host. At this point, the application should read all the data available and process it. The ReceiveCount property can be used to find out how many total bytes of data are waiting in the receive buffer.

If the ReceiveLen property is set to a value less than the ReceiveCount property, then the application may have to read the Receive property more than once to read all available data. In general, the application should get data from the receive buffer until no more data is available. This condition can be detected by checking the length of the string to which the Receive property is assigned. As soon as this length is zero, all data has been read.

Since the ReceiveLen property can be changed at any time (even while connected), an application could assign the value of the ReceiveCount property to ReceiveLen. Any subsequent assignment of the Receive property would then copy all available data from the receive buffer into the applications buffer. This approach is not advisable if very large amounts of data are being received.

The Result property will contain the total number of bytes read from the receive buffer. The Result property will contain a value less than or equal to the value specified by the ReceiveLen property.

This property can only be read at run time while a connection is established. There is no default value for this property.

### Example

```
Dim Message As String
```

```
Message = Socket.Receive
```

## 2.24 ReceiveCount

### Summary

Number of bytes waiting in receive buffer.

### Description

The ReceiveCount property specifies how many bytes of data are available in the receive buffer. This property is read only and can only be accessed while a connection is established.

Because of the interrupt driven nature of network communications, the value of the ReceiveCount property can change quickly at any time. An application should not rely on this value for an extended period of time without rechecking it.

The receive buffer can be accessed with the Receive property. The number of bytes read from the receive buffer while accessing the Receive property can be set with the ReceiveLen property.

This property can only be read at run time while a connection is established. There is no default value for this property.

### Example

```
If Socket.ReceiveCount > 0 Then
    MsgBox "Data is now available in the Receive buffer", 64, "Sample Program"
End If
```

## 2.25 ReceiveLen

### Summary

Maximum number of bytes to read from receive buffer.

### Description

The ReceiveLen property determines the maximum number of bytes of data that are read from the receive buffer when an application accesses the Receive property.

The default value of 100 should be suitable for most applications, but any value other than zero is legal. Setting the ReceiveLen property value to one will cause the application to read one byte of data at a time. If an application wants to make sure that it reads the total number of bytes available, then it should set the value of the ReceiveLen property equal to the value of the ReceiveCount property before accessing the Receive property.

The receive buffer can be accessed with the Receive property. The total number of bytes waiting in the receive buffer can be determined by checking the ReceiveCount property.

This property can be changed at any time. The default value for this property is 100 bytes.

### Example

Socket.**ReceiveLen** = 25

## 2.26 ReceiveOOB

### Summary

Receive buffer for out-of-band data.

### Description

The ReceiveOOB property is used to access any out-of-band data that has been sent by the remote machine. This property can only be accessed by assigning its value to another string while a connection is established. Whenever the property is assigned to a string, as many bytes of out-of-band data as possible up to the number specified by the ReceiveLen property are copied from the out-of-band receive buffer into that string. Out-of-band data can be received only over a TCP connection.

Usually, this property is read during an OnReceiveOOB event. This event informs the application that more out-of-band data has arrived from the connected host. At this point, the application should read all available out-of-band data and process it. The OnReceiveOOB event will only occur if the Option property has previously been set to OPTION\_RECV\_OOB\_ON.

The Result property will contain the total number of bytes read from the out-of-band receive buffer. The Result property will contain a value less than or equal to the value specified by the ReceiveLen property.

This property can only be read at run time while a connection is established. There is no default value for this property.

### Example

```
Dim UrgData As String
```

```
UrgData = Socket.ReceiveOOB
```

## 2.27 ReceiveSize

### Summary

Size in bytes of receive buffer.

### Description

The ReceiveSize property is used to specify the size of the receive buffer of a Socket control. Applications which will receive large amounts of data can increase the size of the receive buffer. By increasing the receive buffer size, the ActiveX control may be able to increase throughput for some applications.

The SendSize property can be used to adjust the size of the send buffer for applications which will send large amounts of data. The ReceiveSize and the SendSize properties are independent of each other.

Most applications will not need to modify this property. Most protocol stacks do not allow the send and receive buffer sizes to be less than 1024 bytes.

This property can be changed at any time before a connection is established (by setting the Action property to ACTION\_CONNECT) or the socket is put into listening mode (by setting the Action property to ACTION\_LISTEN). The default value of this property is 1024 bytes.

### Example

```
Socket.ReceiveSize = 4096
```

## 2.28 RemotePort

### Summary

Remote port to be used for next connection.

### Description

The RemotePort property specifies the port on the remote machine with which a connection should be established. This property is used for client TCP and for UDP sockets. Client TCP sockets must set the RemotePort property just once before a connection is established. Once connected, the remote port remains the same for the entire session. Server TCP sockets do not know ahead of time to which remote port they will be connecting until a connection request arrives.

The RemotePort property must be set before a connection is established with a client TCP socket (by setting the Action property to ACTION\_CONNECT). The Windows Socket ActiveX control does not verify the setting of the RemotePort property and any value (except 0) is therefore legal.

UDP sockets are not connected to any one particular remote port, but are instead able to send and receive messages to and from different remote ports on multiple hosts. Therefore, the RemotePort property is ignored while connecting a UDP socket. Instead, the RemotePort property can be set at any time to change the destination of any subsequent outgoing messages.

Most TCP/IP implementations include a SERVICES file which lists the port numbers for many commonly available TCP and UDP services.

This property can be changed at any time except after a connection has been established with a client TCP socket (by setting the Action property to ACTION\_CONNECT). There is no default value for this property.

### Example

```
Socket.HostName = "127.43.101.12"  
Socket.RemotePort = 13  
Socket.HostConvert = CONVERT_NAME_TO_ADDRESS  
Socket.Action = ACTION_CONNECT
```

## 2.29 Result

### Summary

Number of bytes sent or received.

### Description

The Result property contains the number of bytes that were sent during a send operation and the number of bytes read during a receive operation. This property can only be read by the application and cannot be changed.

Data is sent by assigning a string to the Send property. Care should be taken not to exceed the buffer and transport capabilities of the underlying protocol stack. When using the TCP protocol, in general no more than 512 bytes of data should be assigned to the Send property at any one time. When using the UDP protocol, then no more bytes than the maximum packet size should be assigned to the Send property at any one time. The maximum packet size over Ethernet allows for up to 1500 bytes of data. If the underlying protocol stack is capable of handling fragmented packets, then up to 8000 bytes of data may be sent with every assignment to the Send property.

The Result property must be checked each time after assigning data to the Send property or after accessing the Receive property. If an error occurs during the send operation, then the Result property will contain the number of bytes that were successfully sent before the error occurred. Hence, this property may be used to verify whether all of the data or only a portion has been sent to the remote host. The Error property will contain any error code returned by the protocol stack.

Data is received by accessing the Receive property. This property can only be accessed by assigning its value to another string while a connection is established. Whenever the property is assigned to a string, as many bytes of data as possible up to the number specified by the ReceiveLen property are copied from the receive buffer into that string.

The Result property must be checked each time the Receive property is accessed to read the data sent by the remote machine. The ReceiveLen property specifies the maximum number of bytes that can be read from the receive buffer and the Result property specifies the exact number of bytes that were read from the receive buffer. The Result property will always contain a value less than or equal to the value specified by the ReceiveLen property.

This property can only be read at run time while a connection is established. There is no default value for this property.

### Example

```
Socket.Send = "This is a test"  
If Socket.Result = 14 Then  
    MsgBox "Message sent successfully", 64, "Sample Program"  
End If
```

## 2.30 Send

### Summary

Send buffer.

### Description

The Send property is used to transfer data to the remote machine over an established connection. Data is sent by assigning a string to this property. Care should be taken not to exceed the buffer and transport capabilities of the underlying protocol stack.

When using the TCP protocol, then in general no more than 512 bytes of data should be assigned to this property at any one time. Most TCP/IP stacks are able to transfer multiple 512 byte chunks of data in quick succession. Because of the stream nature of the TCP protocol, every assignment of data to the Send property does not necessarily correspond to a packet being sent over the network. Data may be split up over more than one packet or several buffers may be combined and sent as a single packet.

When using the UDP protocol, then no more bytes than the maximum packet size should be assigned to this property at any one time. Because of the datagram nature of the UDP protocol, each time data is assigned to the Send property, a complete packet will be sent over the network. The maximum packet size over Ethernet allows for up to 1500 bytes of data. If the underlying protocol stack is capable of handling fragmented packets, then up to 8000 bytes of data may be sent with every assignment to the Send property.

The Result property will contain the total number of bytes sent. If no error occurred, then the Result property will be equal to the number of bytes assigned to the Send property.

Incoming data can be read by accessing the Receive property in response to OnReceive events. The ReceiveCount property can be used to check how much data (if any) is available to be read.

This property can only be written to at run time while a connection is established. There is no default value for this property.

### Example

```
Socket.Send = "This is a test"
```

## 2.31 SendOOB

### Summary

Send buffer for out-of-band data.

### Description

The SendOOB property is used to transfer out-of-band data to the remote machine over an established TCP connection. Out-of-band data is sent by assigning a string to this property. Care should be taken not to exceed the buffer and transport capabilities of the underlying protocol stack. Out-of-band data can be sent only over a TCP connection.

Most TCP/IP stacks are able to transfer multiple 512 byte chunks of data in quick succession. Because of the stream nature of the TCP protocol, every assignment of data to the SendOOB property does not necessarily correspond to a packet being sent over the network. Data may be split up over more than one packet or several buffers may be combined and sent as a single packet.

The Result property will contain the total number of bytes sent. If no error occurred, then the Result property will be equal to the number of bytes assigned to the SendOOB property.

Incoming out-of-band data can be read by accessing the ReceiveOOB property in response to OnReceiveOOB events.

This property can only be written to at run time while a connection is established. There is no default value for this property.

### Example

```
' UrgData contains out-of-band data to be sent to the remote machine.  
Socket.SendOOB = UrgData
```

## 2.32 SendSize

### Summary

Size in bytes of send buffer.

### Description

The `SendSize` property is used to specify the size of the send buffer of a Socket control. Applications which will send large amounts of data can increase the size of the send buffer. By increasing the send buffer size, the ActiveX control may be able to increase throughput for some applications.

The `ReceiveSize` property can be used to adjust the size of the receive buffer for applications which will receive large amounts of data. The `SendSize` and the `ReceiveSize` properties are independent of each other.

Most applications will not need to modify this property. Most protocol stacks do not allow the send and receive buffer sizes to be less than 1024 bytes.

This property can be changed at any time before a connection is established (by setting the Action property to `ACTION_CONNECT`) or the socket is put into listening mode (by setting the Action property to `ACTION_LISTEN`). The default value of this property is 1024 bytes.

### Example

```
Socket.SendSize = 4096
```

## 2.33 SendTimeout

### Summary

Time-out when sending data.

### Description

The SendTimeout property specifies in seconds how long the Windows Socket ActiveX control must wait for the underlying protocol stack to buffer the data in the Send property. The default value is 20 seconds.

Usually, this property should be preset during design time to an appropriate value. However, an application can change the value of the SendTimeout property at any time to suit the particular environment.

If the send operation times out, then an OnError event is fired notifying the application that not all of the data was transferred. The Result property will contain the total number of bytes of data sent before the error occurred and the Error property will contain the error code returned by the underlying protocol stack. If no error occurred, then the Result property will be equal to the number of bytes of data assigned to the Send property.

Most applications will not need to modify this property.

This property can be changed at any time. The default value of this property is 20 seconds.

### Example

```
Dim Message As String
```

```
Message = "This is a test"
```

```
Socket.SendTimeout = 15
```

```
Socket.Send = Message
```

## 2.34 ServiceName

### Summary

Official service name.

### Description

The ServiceName property specifies the official name of a service. A service is identified by the protocol name and the port on which the service is available. This property must be set before setting the Convert property to SVC\_NAME\_TO\_PORT.

If the Convert property is set to either SVC\_NAME\_TO\_PORT or SVC\_PORT\_TO\_NAME, then the name of the protocol to use when contacting the service must also be specified in the ProtocolName property.

The Windows Socket ActiveX control can look up either the service port given the name or the service name given the port. To convert the service name to the port, set the ServiceName property to the name of the service and then set the Convert property to SVC\_NAME\_TO\_PORT. To retrieve the service name, set the ServicePort property to the port of the service and then set the Convert property to SVC\_PORT\_TO\_NAME.

This property can only be changed at run time. There is no default value for this property.

### Example

```
Socket.ServiceName = "daytime"  
Socket.ProtocolName = "tcp"  
Socket.Convert = SVC_NAME_TO_PORT  
' Socket.ServicePort will now contain the port number of the "daytime" service for "tcp" protocol.
```

## 2.35 ServicePort

### Summary

Port number on which service is available.

### Description

The ServicePort property specifies the port on which a particular service is available. A service is identified by the protocol name and the service port. This property must be set before setting the Convert property to SVC\_PORT\_TO\_NAME.

If the Convert property is set to either SVC\_NAME\_TO\_PORT or SVC\_PORT\_TO\_NAME, then the name of the protocol to use when contacting the service must also be specified in the ProtocolName property.

The Windows Socket ActiveX control can look up either the service port given the name or the service name given the port. To convert the service name to the port, set the ServiceName property to the name of the service and then set the Convert property to SVC\_NAME\_TO\_PORT. To retrieve the service name, set the ServicePort property to the port of the service and then set the Convert property to SVC\_PORT\_TO\_NAME.

This property can only be changed at run time. There is no default value for this property.

### Example

```
Socket.ServicePort = 13
Socket.ProtocolName = "tcp"
Socket.Convert = SVC_PORT_TO_NAME
' Socket.ServiceName will now contain "daytime".
```

## 2.36 SourceAddress

### Summary

Source internet address of last received datagram or connection request.

### Description

The SourceAddress property is used to determine the source internet address of the last data read by an application. Data is read by accessing the Receive property usually in response to OnReceive events.

Server TCP sockets which are listening for incoming connection requests, will receive an OnAccept event when a connection request arrives. Once the connection is established (using ACTION\_ACCEPT), an OnConnect event will be fired. The SourceAddress property will contain the internet address of the remote machine once the connection has been established. The application can check this address to determine if it should accept the connection request.

For client TCP sockets, the SourceAddress property will be identical to the HostAddress property which specifies the remote machine to which the socket is connected. TCP sockets can only be connected to one host during one session.

The SourcePort property can be accessed to check the source port of the last data read or the source port of the current connection request.

This property can only be read at run time while connected or during an OnAccept event. There is no default value for this property.

### Example

```
Sub Socket_OnConnect ()
  If Socket.HostAddress <> Socket.SourceAddress Then
    Socket.Action = ACTION_DISCONNECT
  End If
End Sub
```

## 2.37 SourcePort

### Summary

Source port of last received datagram or connection request.

### Description

The SourcePort property is used to determine the source port of the last data read by an application. Data is read by accessing the Receive property usually in response to OnReceive events.

Server TCP sockets which are listening for incoming connection requests, will receive an OnAccept event when a connection request arrives. Once the connection is established (using ACTION\_ACCEPT), an OnConnect event will be fired. The SourcePort property will contain the port of the remote machine once the connection has been established. The application can check this port to determine if it should accept the connection request.

For client TCP sockets, the SourcePort property will be identical to the RemotePort property which specifies the remote port to which the socket is connected. TCP sockets can only be connected to one port during one session.

The SourceAddress property can be accessed to check the source internet address of the last data read or the source internet address of the current connection request.

This property can only be read at run time while connected or during an OnAccept event. There is no default value for this property.

### Example

```
Sub Socket_OnConnect ()
    If Socket.RemotePort <> Socket.SourcePort Then
        Socket.Action = ACTION_DISCONNECT
    End If
End Sub
```

### 3 Windows Sockets Events

#### 3.1 OnAccept

##### Summary

Connection request has been received and is ready to be accepted.

##### Description

The OnAccept event occurs when a connection request has been received from a remote machine. This event will only occur on TCP sockets which have been set to listening mode by setting the Action property to ACTION\_LISTEN (or by calling the Listen method).

The SourceAddress and SourcePort properties can be used to determine from which remote machine and port the connection request originated once the connection has been established.

If an application decides not to accept the request, then this event can be ignored. Once an application decides to accept the incoming connection request, it must set the Action property to ACTION\_ACCEPT (or call the Accept method). If the connection can be established, the OnConnect event will occur before the next line of code is reached.

If a connection request is not accepted, then the client may resend the request. In this case, the application will receive more than one OnAccept event.

While handling the OnAccept event, an application should not perform tasks which have the potential of requiring a lot of time to complete, such as generating a message box. A substantial delay could cause the underlying protocol to reject the connection request.

##### Example

```
Sub Socket_OnAccept ()  
    Socket.Action = ACTION_ACCEPT  
End Sub
```

## 3.2 OnClose

### Summary

Connection has been closed.

### Description

The OnClose event occurs usually in response to setting the Action property to ACTION\_DISCONNECT (or in response to the Disconnect method). In some cases, the remote machine may close a connection (for example because of a long period of inactivity or because all data has been transferred). This will also trigger an OnClose event. In this case, the application must still set the Action property to ACTION\_DISCONNECT (or it must call the Disconnect method) to free up all the resources allocated for the connection. However, this should not be done during the OnClose event because it might result in an infinite loop.

If this event occurs in response to setting the Action property to ACTION\_DISCONNECT (or in response to the Disconnect method), it will occur before the statement following the assignment of ACTION\_DISCONNECT (or the call to the Disconnect method) to the Action property is reached.

Normally, an application should simply set a flag in response to this event. Then, this flag can be checked directly after the ACTION\_DISCONNECT (or the Disconnect method) action to make sure that the connection was actually terminated.

### Example

```
Sub Socket_OnClose ()  
    If Connected = True Then  
        Connected = False  
        Socket.Action = ACTION_DISCONNECT  
    End If  
End Sub
```

### 3.3 OnConnect

#### Summary

Connection has been established.

#### Description

The OnConnect event occurs in response to setting the Action property to ACTION\_CONNECT (or in response to the Connect method) on a UDP or client TCP socket or in response to setting the Action property to ACTION\_ACCEPT (or in response to the Accept method) on a listening TCP server socket after an OnAccept event occurred.

This event will occur before the statement following the assignment of ACTION\_CONNECT (or the call to the Connect method) or ACTION\_ACCEPT (or the Accept method) to the Action property is reached. For an asynchronous connect, ACTION\_CONNECT (or the Connect method) will return immediately and this event will occur once the connection has been established.

The SourceAddress and SourcePort properties can be used to determine from which remote machine and port the connection request originated.

Normally, an application should simply set a flag in response to this event. Then, this flag can be checked directly after the ACTION\_CONNECT (or the Connect method) or ACTION\_ACCEPT (or the Accept method) operation to make sure that the connection was actually established.

If the connection could not be established, then the OnError event will be called instead of the OnConnect event.

While handling the OnConnect event, an application should not perform tasks which have the potential of requiring a lot of time to complete, such as generating a message box

#### Example

```
Sub Socket_OnConnect ()  
    Connected = True  
End Sub
```

### 3.4 OnError

#### Summary

Local error has occurred.

#### Description

The OnError event occurs when a property is accessed in an illegal way or when a connection cannot be established. The following table describes all possible error codes delivered by this event.

<b>Value</b>	<b>Meaning</b>
ERR_CHANGE_PROTOCOL	Cannot change protocol while connected.
ERR_CHANGE_LOCAL_PORT	Cannot change local port while connected.
ERR_CHANGE_REMOTE_PORT	Cannot change remote port while connected.
ERR_CHANGE_RECV_SIZE	Cannot change receive buffer size while connected.
ERR_CHANGE_SEND_SIZE	Cannot change send buffer size while connected.
ERR_CHANGE_HOST_ADDR	Cannot change host address while connected.
ERR_NO_HOST_ADDR	Remote host address not defined.
ERR_CANNOT_GET_SOCKET	Unable to allocate socket.
ERR_CANNOT_BIND_SOCKET	Unable to bind socket.
ERR_CANNOT_CONNECT_SOCKET	Unable to connect.
ERR_CANNOT_SETUP_SOCKET	Unable to setup socket.
ERR_CANNOT_LISTEN_SOCKET	Unable to listen.
ERR_ALREADY_CONNECTED	Cannot listen on a connected socket.
ERR_ALREADY_LISTENING	Cannot connect a listening socket.
ERR_LISTENING_REMOTE_PORT	Cannot change remote port while listening.
ERR_LISTENING_HOST_ADDR	Cannot change host address while listening.
ERR_NOT_LISTENING	Server is not listening.
ERR_CANNOT_ACCEPT	Unable to accept connection.
ERR_CONNECT_TO_RECV	Connect to receive data.
ERR_CONNECT_TO_RECV_COUNT	Connect to obtain receive count.
ERR_CONNECT_TO_SEND	Connect to send data.
ERR_CANNOT_SEND	Unable to send data.
ERR_NO_OPTION_VALUE	Value of socket option not defined.
ERR_CANNOT_GET_LOCAL	Unable to obtain local IP address.
ERR_CANNOT_RECV_OOB	Unable to receive out-of-band data.
ERR_CANNOT_SEND_OOB	Unable to send out-of-band data.
ERR_INVALID_PROTOCOL	Cannot use specified protocol for the specified action.
ERR_FW_SERVER_NOT_DEFINED	Firewall server is not specified.
ERR_FW_PORT_NOT_DEFINED	Firewall port is not defined.
ERR_REMOTE_PORT_NOT_DEFINED	Remote port is not defined.
ERR_HOST_NOT_DEFINED	Remote host is not specified.
ERR_CANNOT_CONNECT	Cannot connect to remote machine.
ERR_CHANGE_FW_PORT	Cannot change firewall port while connected.

The following describes each error in more detail.

**ERR\_CHANGE\_PROTOCOL**

Protocol cannot be changed while connected or while listening.

**ERR\_CHANGE\_LOCAL\_PORT**

Local port cannot be changed while connected or while listening.

**ERR\_CHANGE\_REMOTE\_PORT**

Remote port cannot be changed while connected on TCP.

**ERR\_CHANGE\_RECV\_SIZE**

Size of receive buffer cannot be changed while connected or while listening.

**ERR\_CHANGE\_SEND\_SIZE**

Size of send buffer cannot be changed while connected or while listening.

**ERR\_CHANGE\_HOST\_ADDR**

Remote host address cannot be changed while connected on TCP.

**ERR\_NO\_HOST\_ADDR**

Internet address of remote host not defined. The HostConvert property can be used to convert a host name to an internet address.

**ERR\_CANNOT\_GET\_SOCKET**

Unable to allocate socket.

**ERR\_CANNOT\_BIND\_SOCKET**

Unable to bind socket to local address.

**ERR\_CANNOT\_CONNECT\_SOCKET**

Unable to connect to remote host.

**ERR\_CANNOT\_SETUP\_SOCKET**

Unable to set up socket.

**ERR\_CANNOT\_LISTEN\_SOCKET**

Unable to listen on socket.

**ERR\_ALREADY\_CONNECTED**

Cannot listen on a connected socket. The Action property was set to ACTION\_LISTEN (or the Listen method was called) while a connection was established.

**ERR\_ALREADY\_LISTENING**

Cannot connect a listening socket. The Action property was set to ACTION\_CONNECT (or call the Connect method was called) when the socket was in the listen mode.

**ERR\_LISTENING\_REMOTE\_PORT**

Remote port cannot be changed while listening on TCP.

**ERR\_LISTENING\_HOST\_ADDR**

Remote host address cannot be changed while listening on TCP.

**ERR\_NOT\_LISTENING**

Cannot accept on a socket that is not listening. Set the Action property to ACTION\_LISTEN (or the Listen method) before accepting connections.

**ERR\_CANNOT\_ACCEPT**

Unable to accept incoming TCP connection.

**ERR\_CONNECT\_TO\_RECV**

Must be connected before accessing receive buffer.

**ERR\_CONNECT\_TO\_RECV\_COUNT**

Must be connected before accessing receive count.

**ERR\_CONNECT\_TO\_SEND**

Must be connected before accessing send buffer.

**ERR\_CANNOT\_SEND**

Unable to send data to remote host.

**ERR\_NO\_OPTION\_VALUE**

Value of socket option not defined. Set the OptionValue property to the time-out for linger before setting the Option property to OPTION\_LINGER\_ON (also see Linger method).

**ERR\_CANNOT\_GET\_LOCAL**

Unable to obtain the local host's name or dotted decimal internet address.

**ERR\_CANNOT\_RECV\_OOB**

Unable to receive out-of-band data from remote host. Out-of-band data can only be received over TCP connections. Make sure that the Protocol property has been set to PROTOCOL\_TCP.

**ERR\_CANNOT\_SEND\_OOB**

Unable to send out-of-band data to remote host. Out-of-band data can only be sent over TCP connections. Make sure that the Protocol property has been set to PROTOCOL\_TCP.

**ERR\_INVALID\_PROTOCOL**

Unable to connect to remote host via a firewall. Connections to remote host via a firewall can only be established with the TCP transport protocol. Make sure that the Protocol property has been set to PROTOCOL\_TCP.

**ERR\_FW\_SERVER\_NOT\_DEFINED**

Firewall server has not been initialized. Connections to remote host via a firewall can only be established if the firewall server is defined. Make sure that the FirewallServer property has been set to a valid firewall server name or IP address.

**ERR\_FW\_PORT\_NOT\_DEFINED**

Firewall port has not been initialized to a non-zero value. Connections to remote host via a firewall can only be established if the firewall port is defined. Make sure that the FirewallPort property has been set to a valid firewall server port.

**ERR\_FW\_REMOTE\_PORT\_NOT\_DEFINED**

Remote port has not been initialized to a non-zero value. Connections to remote host via a firewall can only be established if the remote port is defined. Make sure that the RemotePort property has been set to a port on which the remote machine is listening for connections

**ERR\_HOST\_NOT\_DEFINED**

Remote host has not been initialized. Connections to remote host via a firewall can only be established if the remote host is defined. Make sure that the HostName property has been set to a valid remote host name or IP address.

**ERR\_CANNOT\_CONNECT**

Unable to connect to the remote host via a firewall. Make sure that the Firewall server is running and all necessary information, such as firewall server name, firewall port, remote host and remote port are correct.

**ERR\_CHANGE\_FW\_PORT**

Firewall port cannot be changed while connected.

**Example**

```
Sub Socket_OnError (ErrorCode As Integer)
  If ErrorCode = ERR_CANNOT_CONNECT_SOCKET Then
    MsgBox "Unable to connect to remote host", 64, "Sample Program"
  End If
End Sub
```

### 3.5 OnReceive

#### Summary

More data has been added to the receive buffer.

#### Description

The OnReceive event occurs when more data arrives from the remote machine over a connected socket. The event does not actually deliver the data to the application but instead expects that the application will access the Receive property to obtain all data waiting in the receive buffer.

Each OnReceive event usually corresponds to one packet of data arriving for the socket. Depending on the underlying protocol stack, OnReceive events may also be fired after each access to the receive buffer through the Receive property if not all available data has been read. In some of these environments, for example Windows Sockets, no duplicate OnReceive events will be fired. This means that only one OnReceive event will occur until the Receive property has been accessed even if more data has arrived.

An application must be able to deal with OnReceive events which occur even though no data is available to receive from the socket. This condition may occur if during one event the application reads all the available data including data associated with a subsequent event.

The ReceiveCount property can be checked to determine how many bytes of data are available to read. Because of the interrupt driven nature of network communications, the value of the ReceiveCount property can change quickly at any time. An application should not rely on this value for an extended period of time without rechecking it.

The ReceiveLen property specifies how many bytes of data will be obtained from the receive buffer every time the Receive property is accessed.

The Result property will contain the total number of bytes read from the receive buffer. The Result property will contain a value less than or equal to the value specified by the ReceiveLen property.

An application does not have to rely on the OnReceive event to notify it of incoming data. Instead of or in addition to this event, the application may check the value of the ReceiveCount property to find out if any data is available.

For more information on how to receive data, please check the reference pages of the Receive, ReceiveCount, ReceiveLen and Result properties.

While handling the OnReceive event, an application should not perform tasks which have the potential of requiring a lot of time to complete, such as generating a message box. A substantial delay could cause the application to not receive subsequent OnReceive events.

#### Example

```
Sub Socket_OnReceive ()  
    Dim Message As String  
  
    Message = Socket.Receive  
End Sub
```

### 3.6 OnReceiveOOB

#### Summary

More out-of-band data has been received.

#### Description

The OnReceiveOOB event occurs when more out-of-band data arrives from the remote machine over a connected TCP socket. The event does not actually deliver the data to the application but instead expects that the application will access the ReceiveOOB property to obtain the out-of-band data waiting in the out-of-band receive buffer. This event will be fired only if the Option property has been set to OPTION\_RECV\_OOB\_ON. This option enables notification on arrival of out-of-band data.

Each OnReceiveOOB event usually corresponds to one packet of out-of-band data arriving for the socket. Depending on the underlying protocol stack, OnReceiveOOB events may also be fired after each access of the out-of-band receive buffer through the ReceiveOOB property if not all available out-of-band data has been read. In some of these environments, for example Windows Sockets, no duplicate OnReceiveOOB events will be fired. This means that only one OnReceiveOOB event will occur until the ReceiveOOB property has been accessed even if more out-of-band data has arrived.

An application must be able to deal with OnReceiveOOB events which occur even though no out-of-band data is available on the socket. This condition may occur if during one event the application reads all the available out-of-band data including out-of-band data associated with a subsequent event.

The ReceiveLen property specifies how many bytes of out-of-band data will be obtained from the out-of-band receive buffer every time the ReceiveOOB property is accessed.

The Result property will contain the total number of bytes read from the out-of-band receive buffer. The Result property will contain a value less than or equal to the value specified by the ReceiveLen property.

For more information on how to receive out-of-band data, please check the reference pages of the Option, ReceiveOOB, ReceiveLen and Result properties.

While handling the OnReceiveOOB event, an application should not perform tasks which have the potential of requiring a lot of time to complete, such as generating a message box. A substantial delay could cause the application to not receive subsequent OnReceiveOOB events.

#### Example

```
Sub Socket_OnReceiveOOB ()  
    Dim UrgData As String  
  
    UrgData = Socket.ReceiveOOB  
End Sub
```

## 4. Windows Sockets Methods

### 4.1 Accept

#### Summary

Accept an incoming connection request.

#### Syntax

**Boolean Accept ()**

#### Description

Whenever a client sends a connection request over the network to the port on which the server is listening, an OnAccept event is fired. In response to the OnAccept event, the application must call the Accept method to establish a connection with the client. A server TCP socket is created by setting the Action property to ACTION\_LISTEN or by calling the Listen method.

The Accept method takes no parameters and returns a boolean. If an incoming connection request can be successfully established, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value.

If the connection can be established, then the OnConnect event will occur before the next line of code is reached. Once a connection has been established, data can be sent with the Send property and available data can be received with the Receive property.

Calling this method is equivalent to setting the Action property to ACTION\_ACCEPT.

#### Example

```
Result = Socket.Accept ()
If Result = False Then
    MsgBox "Cannot accept request", 64, "Sample Program"
Exit Sub
End If
```

## 4.2 AddrToDecimal

### Summary

Convert host address to dotted decimal.

### Syntax

```
String AddrToDecimal (HostAddress)  
    HostAddress    Long
```

### Description

The AddrToDecimal method converts the specified internet address passed as a parameter into the dotted decimal address of the host.

The AddrToDecimal method takes a host address (*HostAddress*) as its parameter and returns a string. This method converts the host address to its corresponding dotted decimal address. The result of this lookup will be returned as a string.

If the *HostAddress* can be successfully converted to its dotted decimal address, then the method returns the dotted decimal address; otherwise, it returns an empty string. If the given host address is 0, then the HostName property will also be cleared. The application should ensure that the method was successfully executed by checking the return value.

Calling this method is equivalent to setting the HostConvert property to CONVERT\_ADDRESS\_TO\_DECIMAL.

### Example

```
Name = Socket.AddrToDecimal (Socket.HostAddress)  
If Name = "" Then  
    MsgBox "Cannot convert host address to dotted decimal format", 64, "Sample Program"  
End If
```

### 4.3 AddrToName

#### Summary

Convert host address to host name.

#### Syntax

```
String AddrToName (HostAddress)  
HostAddress Long
```

#### Description

The AddrToName method converts the internet address passed as parameter into the name of the host. The control first checks if the specified internet address is defined in the local hosts data base. If the address is not found, then the control will send queries over the network to a name server.

The AddrToName method takes a host address long as its parameter and returns a string. This method takes the host address long parameter and converts to the corresponding host name. The result of this lookup will be returned as a string.

If the host address can be successfully converted to its host name, then the method returns the host name; otherwise, it returns an empty string. If the given host address is 0, then the HostName property will also be cleared. The application should ensure that the method was successfully executed by checking the return value.

Calling this method is equivalent to setting the HostConvert property to CONVERT\_ADDRESS\_TO\_NAME.

#### Example

```
HostName = Socket.AddrToName (Socket.HostAddress)  
If HostName = "" Then  
    MsgBox "Cannot convert the host address to a host name", 64, "Sample Program"  
End If
```

## 4.4 AsyncConnect

### Summary

Use or not use asynchronous connect.

### Syntax

**Boolean AsyncConnect (*Enable*)**  
*Enable* Boolean

### Description

The AsyncConnect method establishes a connection in asynchronous mode or not asynchronous mode.

The AsyncConnect method takes a enable flag (*Enable*) as its parameter and returns a boolean. If asynchronous connect can be successfully enabled or disabled, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value.

This method has to be called before calling the Connect method or setting the Action property to ACTION\_CONNECT. If the AsyncConnect is called with *Enable* set to True, then this will cause the Connect or ACTION\_CONNECT to return immediately without waiting for the connection to be established. Later, once the connection is established, an OnConnect event will be fired. To disable asynchronous connect, call the AsyncConnect method with *Enable* set to False before establishing a connection using ACTION\_CONNECT or calling the Connect method. Connections are **not** established in asynchronous mode by default.

Calling this method is equivalent to setting the Option property to OPTION\_ASYNC\_CONNECT\_ON or OPTION\_ASYNC\_CONNECT\_OFF.

This method can be called at any time before a connection is established.

### Example

```
Result = Socket.AsyncConnect (True)
If Result = False Then
    MsgBox "Cannot enable asynchronous connect", 64, "Sample Program"
End If
```

## 4.5 Broadcast

### Summary

Allow or not allow transmission of broadcast messages.

### Syntax

**Boolean Broadcast (*Enable*)**  
*Enable*          Boolean

### Description

The Broadcast method is only available for UDP connections.

The Broadcast method takes a enable flag (*Enable*) as its parameter and returns a boolean. If broadcast messages are successfully enabled or disabled, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value.

To allow transmission of broadcast messages on such a socket, the Broadcast method must be called with *Enable* set to True. When calling the Broadcast method with *Enable* set to False, broadcast messages can no longer be transmitted on the socket. This method can be called even after a connection has been established. By default the Windows Socket ActiveX control enables broadcast for UDP connections.

Calling this method is equivalent to setting the Option property to OPTION\_BROADCAST\_ON or OPTION\_BROADCAST\_OFF.

This method can be called at any time before or after a connection is established.

### Example

```
Result = Socket.Broadcast (True)
If Result = False Then
    MsgBox "Cannot enable broadcast messages", 64, "Sample Program"
End If
```

## 4.6 Connect

### Summary

Establish connection.

### Syntax

#### **Boolean Connect (*Protocol*, *HostAddress*, *RemotePort*)**

<i>Protocol</i>	Integer
<i>HostAddress</i>	Long
<i>RemotePort</i>	Integer

### Description

The Connect method establishes a connection.

The Connect method takes a protocol (*Protocol*), a host address (*HostAddress*), a remote port (*RemotePort*) as its parameters and returns a boolean. The protocol must be set to select the TCP or UDP transport protocol. The host address long must be set to the internet address of the remote machine. The HostConvert property or the NameToAddr method can be used to convert a host name (or a host address in dotted decimal notation) into an internet address. The remote port integer must be set to the port on which the remote service to connect to is running. For a server TCP socket, the remote port is not known until a connection request from a remote machine has been accepted. For a UDP socket, the remote port integer is not used while connecting and can be changed at any time after calling the Connect method.

If a connection is successfully established, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value.

If the connection can be established, then the OnConnect event will be fired. If the connection cannot be established, then the OnError event will be fired. These events will occur before the next statement (i.e. the statement following the call to the Connect method) is executed. The application should set a flag in the OnConnect and OnError events, so that it can determine if the session has been established or not. In addition, the application may want to display an error message in the OnError event to inform the user that the connection has not been established. Please check the reference page of the OnError event for a complete listing of error codes.

If the application wishes to establish connection using an asynchronous connect operation, the Option property must be set to OPTION\_ASYNC\_CONNECT\_ON or call the AsyncConnect method before calling the Connect method. Connect does not wait until connection is established but returns immediately after issuing a connection request. If the connection can be established, the OnConnect event will be fired; otherwise, an OnError event will occur. Check the Error property to determine the error code returned by the underlying protocol stack.

Calling this method is equivalent to setting the Action property to ACTION\_CONNECT.

### Example

```
HostAddress = Socket.NameToAddr ("speedy.distinct.com")
Result = Socket.Connect (PROTOCOL_TCP, HostAddress, 13)
If Result = False Then
    MsgBox "Unable to connect", 64, "Sample Program"
End If
```

## 4.7 Disconnect

### Summary

Close connection

### Syntax

**Boolean Disconnect ()**

### Description

Once a connection is no longer needed, the session can be terminated by calling the Disconnect method. This method closes the connection that was established using the Connect or the FwConnect method. An application must close all connections it has created before it quits.

The Disconnect method takes no parameters and returns a boolean. If a connection is successfully terminated, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value.

Calling this method is equivalent to setting the Action property to ACTION\_DISCONNECT.

### Example

```
Result = Socket.Disconnect ()  
If Result = False Then  
    MsgBox "Unable to disconnect from server", 64, "Sample Program"  
End If
```

## 4.8 FwConnect

### Summary

Establish connection via a firewall.

### Syntax

**Boolean FwConnect (*FwServer*, *FwPort*, *DestHost*, *DestPort*)**

<i>FwServer</i>	String
<i>FwPort</i>	Integer
<i>DestHost</i>	String
<i>DestPort</i>	Integer

### Description

The FwConnect method establishes a connection via a firewall.

If the local machine is located on a different subnet than the remote machine and the only form of communication between these two machines is through a firewall gateway, then the built-in firewall support of the Windows Socket ActiveX control can be used to establish a session.

The FwConnect method takes a firewall server name (*FwServer*), a firewall port (*FwPort*), a remote host (*DestHost*), and a remote port (*DestPort*) as its parameters and returns a boolean. The protocol must be set to select the TCP transport protocol. The firewall server string must be set to the name or IP address of the firewall through which connection is to be established. The firewall port integer must be set to the port on which the firewall server is listening for connections. The destination host string must be set to the name or IP address of the remote machine. If the *DestHost* is the internet address then the *FwAddrType* property must be set to FW\_ADDR\_IP4 and if it is a machine name then the *FwAddrType* property must be set to FW\_ADDR\_DNS, by default the *FwAddrType* property has the value FW\_ADDR\_IP4. The destination port integer must be set to the port on which the remote service to connect to is running.

The application can also set the *FwAuthMethods* property if it wants to specify any authentication that needs to be negotiated before an actual connection is established. Only the Username/Password authentication is currently supported. If Username/Password authentication is specified the *FwUsername* and *FwPassword* should contain a valid username and password respectively. An authentication is only possible if the SOCKS version is 5; version 4 SOCKS server does not support any authentication protocols.

The version of the SOCKS server can be specified in the *FwSocksVer* property. If the version is unknown then the application can set the *FwSocksVer* Property to FW\_VERSION4 and FW\_VERSION5 and the Distinct Windows Socket ActiveX control will automatically detect the SOCKS version and connect appropriately.

If a connection is successfully established, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value.

If the connection can be established, then the OnConnect event will be fired. If the connection cannot be established, then the OnError event will be fired. These events will occur before the next statement (i.e. the statement following the call to the FwConnect method) is executed. The application should set a flag in the OnConnect and OnError events, so that it can determine if the session has been established or not. In addition, the application may want to display an error message in the OnError event to inform the user that the connection has not been established. Please check the reference page of the OnError event for a complete listing of error codes.

Calling this method is equivalent to setting the Action property to ACTION\_FW\_CONNECT.

**Example**

```
Result = Socket.FwConnect ("sparky.distinct.com", 1080, "speedy.distinct.com", 13)
If Result = False Then
    MsgBox "Unable to connect", 64, "Sample Program"
End If
```

## 4.9 KeepAlive

### Summary

Send or not send keep-alive TCP packets.

### Syntax

**Boolean KeepAlive (*Enable*)**  
*Enable* Boolean

### Description

An application may request the underlying protocol stack to enable the use of keep-alive packets on TCP connections (provided the protocol stack supports the use of keep-alives) by calling the KeepAlive method with *Enable* set to True passed. The send and time-out intervals must be specified as required by the protocol stack in use. To disable keep-alive probes, call this method with *Enable* set to False (this is the default setting).

The KeepAlive method takes a enable flag (*Enable*) as its parameter and returns a boolean. If the keep-alive probes can be successfully enabled or disabled, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value.

Calling this method is equivalent to setting the Option property to OPTION\_KEEPALIVE\_ON or OPTION\_KEEPALIVE\_OFF.

This method can be called at any time before or after a connection is established.

### Example

```
Result = Socket.KeepAlive (True)
If Result = False Then
    MsgBox "Cannot enable keep-alive probes", 64, "Sample Program"
    Exit Sub
End If
```

## 4.10 Linger

### Summary

Set or not set linger time-out for close socket operation.

### Syntax

#### Boolean Linger (*Enable*, *OptionValue*)

*Enable*            Boolean

*OptionValue*    String

### Description

The Linger method controls the behavior of the socket while disconnecting.

The Linger method takes a enable flag (*Enable*) and an timeout value (*OptionValue*) as its parameters and returns a boolean. If linger time-out can be successfully enabled or disabled, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value.

If the method is called with *Enable* set to True, then the socket is not closed until all data buffered in the send buffer is transmitted or the given time-out has been exceeded. The time-out value in seconds must be specified as the *OptionValue* parameter. If the *OptionValue* parameter is set to 0 with linger enabled, then the connection will be reset. If the method is called with *Enable* set to False, then the underlying protocol stack will close without waiting for buffered data to be sent. The Windows Sockets ActiveX control by default enables linger with a three second time-out.

Calling this method is equivalent to setting the Option property to OPTION\_LINGER\_ON or OPTION\_LINGER\_OFF.

This method can be called at any time before or after a connection is established.

### Example

```
Result = Socket.Linger (True, "5")
If Result = False Then
    MsgBox "Cannot set linger time-out", 64, "Sample Program"
End If
```

## 4.11 Listen

### Summary

Listen for incoming connection request.

### Syntax

**Boolean Listen (*Protocol*, *LocalPort*)**

*Protocol*           Integer

*LocalPort*         Integer

### Description

The Listen method listens for incoming connection request. A server TCP socket is created by calling this method. No event is generated in response to a successful listen.

The Listen method takes a protocol (*Protocol*) and a local port (*LocalPort*) as its parameters and returns a boolean. The protocol must be set to select the TCP or UDP transport protocol. The *LocalPort* integer must be set to the local port to be associated with the socket. Default local ports (0) cannot be used for server TCP sockets.

If the listen command is successful, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value.

Calling this method is equivalent to setting the Action property to ACTION\_LISTEN.

### Example

```
Result = Socket.Listen (PROTOCOL_TCP, 2000)
If Result = False Then
    MsgBox "Unable to listen", 64, "Sample Program"
End If
```

## 4.12 NameToAddr

### Summary

Convert host name to host address.

### Syntax

```
Long NameToAddr (HostName)  
    HostName    String
```

### Description

The NameToAddr method converts the host name (or the host address specified in dotted decimal notation) into the internet address of the host. If specified host name contains an address in dotted decimal notation (for example, 127.43.101.12), then the control will simply convert the string to an internet address. Otherwise, the control first checks if the specified host is defined in the local hosts data base. If the host is not found, then the control will send queries over the network to a name server.

The NameToAddr method takes a host name (*HostName*) as its parameter and returns a long. This method takes the *HostName* parameter and converts it to the corresponding host address. The result of this lookup will be returned as a long.

If the host name can be successfully converted to its host address, then the method returns the host address long; otherwise, it returns 0. If the given host name cannot be resolved, then the HostAddress property will also be set to 0. The application should ensure that the method was successfully executed by checking the return value.

Calling this method is equivalent to setting the HostConvert property to CONVERT\_NAME\_TO\_ADDRESS.

### Example

```
HostAddress = Socket.NameToAddr ("speedy")  
If HostAddress = 0 Then  
    MsgBox "Cannot convert the specified host name to its host address", 64, "Sample Program"  
End If
```

### 4.13 ProtoNameToNumber

#### Summary

Convert protocol name to protocol number.

#### Syntax

**Integer ProtoNameToNumber (*ProtocolName*)**  
*ProtocolName*     String

#### Description

The ProtoNameToNumber method converts the protocol name into the protocol number associated with the protocol name.

The ProtoNameToNumber method takes a protocol name (*ProtocolName*) as its parameter and returns an integer. This method takes the protocol name parameter and converts it to the associated protocol number. The result of this lookup will be returned as an integer.

If the protocol name can be successfully converted to its protocol number, then the method returns the protocol number; otherwise, it returns 0. If the given protocol name cannot be resolved, then the ProtocolNumber property will also be set to 0. The application should ensure that the method was successfully executed by checking the return value.

Calling this method is equivalent to setting the Convert property to PROTO\_NAME\_TO\_NUMBER.

The Error property can be checked to obtain the error code returned by the underlying protocol stack.

#### Example

```
ProtocolNumber = Socket.ProtoNameToNumber ("tcp")
If ProtocolNumber = 0 Then
    MsgBox "Cannot obtain protocol number", 64, "Sample Program"
End If
```

## 4.14 ProtoNumberToName

### Summary

Convert protocol number to protocol name.

### Syntax

```
String ProtoNumberToName (ProtocolNumber)  
    ProtocolNumber    Integer
```

### Description

The ProtoNumberToName method converts the protocol number into corresponding protocol name.

The ProtoNumberToName method takes a protocol number (*ProtocolNumber*) as its parameter and returns a string. This method takes the protocol number parameter and converts it to the corresponding protocol name. The result of this lookup will be returned as a string.

If the protocol number can be successfully converted to its protocol name, then the method returns the protocol name string; otherwise, it returns an empty string. If the given protocol number cannot be resolved, then the ProtocolName property will also be cleared. The application should ensure that the method was successfully executed by checking the return value.

Calling this method is equivalent to setting the Convert property to PROTO\_NUMBER\_TO\_NAME.

The Error property can be checked to obtain the error code returned by the underlying protocol stack.

### Example

```
ProtocolName = Socket.ProtoNumberToName (1)  
If ProtocolName = "" Then  
    MsgBox "Cannot obtain protocol name", 64, "Sample Program"  
End If
```

## 4.15 ReceiveB

### Summary

Receive binary data.

### Syntax

**Long ReceiveB (*Buffer*)**  
*Buffer* Variant

### Description

The ReceiveB method is used to access binary data that has been sent by the remote machine. Whenever this method is called, as many bytes of data as possible (up to the number specified by the ReceiveLen property) are copied to the buffer parameter passed to it.

The ReceiveB method takes a variant buffer (*Buffer*) as its parameter and returns a long. If operation is successful, then the method fills the variant buffer with data received and returns the number of bytes that was copied; otherwise, it sets the buffer to NULL and returns 0. The application should ensure that the method was successfully executed by checking variant buffer or the return value.

Usually, this method is called during an OnReceive event. This event informs the application that more data has arrived from the connected host. At this point, the application should read all the data available and process it. The ReceiveCount property can be used to find out how many total bytes of data are waiting to be read.

If the ReceiveLen property is set to a value less than the ReceiveCount property, then the application may have to call the ReceiveB method more than once to read all available data. In general, the application should get data from the receive buffer until no more data is available. This condition can be detected by checking the value returned by the ReceiveB method. As soon as this value is zero, all data has been read.

Since the ReceiveLen property can be changed at any time (even while connected), an application could assign the value of the ReceiveCount property to ReceiveLen. Any subsequent call to the ReceiveB method would then copy all available data to the buffer parameter passed to the method. This approach is not advisable if very large amounts of data are being received.

The Result property and the return value of the ReceiveB method will contain the total number of bytes read. The Result property will contain a value less than or equal to the value specified by the ReceiveLen property.

This method can only be read at run time while a connection is established.

### Example

```
Dim Buffer() As Byte
Dim Siz As Long
Dim i As Integer

Siz = Socket.ReceiveB (Buffer)
If Siz > 0 Then
    Open "c:\abc.exe" For Binary Access Write As #1
    For i = 1 To Siz
        Put #1, i, Buffer (i)
    Next i
    Close #1
Else
```

```
    MsgBox "Cannot receive binary data", 64, "Sample Program"  
End If
```

## 4.16 RecvOOB

### Summary

Receive out-of-band data separately or as regular data.

### Syntax

**Boolean RecvOOB (*Enable*)**  
*Enable*            Boolean

### Description

Out-of-band data (or TCP urgent data) can be delivered to the user either separately from regular data or as part of regular data. If an application needs to process out-of-band data apart from regular data, then the RecvOOB method must be called with *Enable* set to True. In this case, the arrival of out-of-band data will trigger the OnReceiveOOB event. The ReceiveOOB property can then be used to access the out-of-band data. To receive out-of-band data as regular data, call this method with *Enable* set to False. By default out-of-band data is received as part of the regular data stream. The out-of-band options are available only for TCP connections.

The RecvOOB method takes a enable flag (*Enable*) as its parameter and returns a boolean. If operation is successful, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value.

Calling this method is equivalent to setting the Option property to OPTION\_RECV\_OOB\_ON or OPTION\_RECV\_OOB\_OFF.

This method can be called at any time before a connection is established.

### Example

```
Result = Socket.RecvOOB (True)
If Result = False Then
    MsgBox "Cannot receive out-of-band data", 64, "Sample Program"
End If
```

## 4.17 ReuseAddr

### Summary

Allow or not allow reuse of local address.

### Syntax

**Boolean ReuseAddr (*Enable*)**  
*Enable*      Boolean

### Description

A socket may by default not be bound to a local address which is already in use. If, however, the application desires to reuse an address, it must call the ReuseAddr method with *Enable* set to True before trying to establish a connection. Since every connection is uniquely identified by the combination of local and remote ports and addresses, two sockets can be bound to the same local port and address as long as the remote port and address are different. To disable the reuse of addresses, call this method with *Enable* set to False before trying to establish a connection (this is the default setting).

The ReuseAddr method takes a enable flag (*Enable*) as its parameter and returns a boolean. If the reuse of local address is successfully enabled or disabled, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value.

Calling this method is equivalent to setting the Option property to OPTION\_REUSEADDR\_ON or OPTION\_REUSEADDR\_OFF.

This method can be called at any time before a connection is established.

### Example

```
Result = Socket.ReuseAddr (True)
If Result = False Then
    MsgBox "Cannot enable reuse of local address", 64, "Sample Program"
End If
```

## 4.18 SendB

### Summary

Send binary data.

### Syntax

#### Boolean SendB (*Buffer*, *Bytes*)

*Buffer* Variant

*Bytes* Long

### Description

The SendB method is used to transfer binary data to the remote machine over an established connection. Care should be taken not to exceed the buffer and transport capabilities of the underlying protocol stack.

The SendB method takes a variant buffer (*Buffer*) and a long byte (*Bytes*) as its parameters and returns a boolean. The variant buffer has to be set to the data to be sent. The bytes parameter has to be set to the number of bytes of data (or the size of the buffer parameter) to be sent. If the operation is successful, then the method returns True; otherwise, it returns False. The application should ensure that the method was successfully executed by checking the return value.

When using the TCP protocol in general no more than 512 bytes of data should be passed to this method at any one time. Most TCP/IP stacks are able to transfer multiple 512 byte chunks of data in quick succession. Because of the stream nature of the TCP protocol, every call to the SendB method with data does not necessarily correspond to a packet being sent over the network. Data may be split up over more than one packet or several buffers may be combined and sent as a single packet.

When using the UDP protocol, no more bytes than the maximum packet size should be passed to this method at any one time. Because of the datagram nature of the UDP protocol, each time data is passed to the SendB method, a complete packet will be sent over the network. The maximum packet size over Ethernet allows for up to 1500 bytes of data. If the underlying protocol stack is capable of handling fragmented packets, then up to 8000 bytes of data may be sent with every call to the SendB method.

The Result property will contain the total number of bytes sent. If no error occurred, then the Result property will be equal to the number of bytes passed to the SendB method.

Incoming data can be read by calling the ReceiveB method in response to OnReceive events. The ReceiveCount property can be used to check how much data (if any) is available to be read.

This method can only be called at run time while a connection is established.

### Example

```
Dim Buffer() As Byte
Dim Bytes As Long
Dim i As Integer

Open "c:\abc.exe" For Binary Access Read As #1
Bytes = FileLen("c:\abc.exe")
For i = 1 To Bytes
    Get #1, , Buffer(i)
Next i
Result = Socket.SendB(Buffer, Bytes)
If Result = False Then
    MsgBox "Cannot send binary data", 64, "Sample Program"
```

End If  
Close #1

## 4.19 SvcNameToPort

### Summary

Convert service name to service port.

### Syntax

**Integer ProtoNameToNumber (*ServiceName*, *ProtocolName*)**

*ServiceName*       String

*ProtocolName*      String

### Description

The SvcNameToPort method converts the service name (associated with the protocol name) into the service port where the service can be contacted. The name of the protocol is used when contacting the service.

The SvcNameToPort method takes a service name (*ServiceName*) and a protocol name (*ProtocolName*) as its parameters and returns an integer. This method takes the service name and the protocol name and converts it to the associated service port. The result of this lookup will be returned as an integer.

If the service name can be successfully converted to its service port, then the method returns the service port; otherwise, it returns 0. If the given service name cannot be resolved, then the ServicePort property will also be set to 0. The application should ensure that the method was successfully executed by checking the return value.

Calling this method is equivalent to setting the Convert property to SVC\_NAME\_TO\_PORT.

The Error property can be checked to obtain the error code returned by the underlying protocol stack.

### Example

```
ServicePort = Socket.SvcNameToPort ("daytime", "tcp")
If ServicePort = 0 Then
    MsgBox "Cannot obtain service port", 64, "Sample Program"
End If
```

## 4.20 SvcPortToName

### Summary

Convert service port to service name.

### Syntax

```
String SvcPortToName (ServicePort, ProtocolName)  
    ServicePort      Integer  
    ProtocolName    String
```

### Description

The SvcPortToName method converts the service port (associated with the protocol name) into the corresponding service name.

The SvcPortToName method takes a service port (*ServicePort*) and a protocol name (*ProtocolName*) as its parameter and returns a string. This method takes the service port and the protocol name and converts it to the corresponding service name. The result of this lookup will be returned as a string.

If the service port can be successfully converted to its service name, then the method returns the service name; otherwise, it returns an empty string. If the given service port cannot be resolved, then the ServiceName property will also be cleared. The application should ensure that the method was successfully executed by checking the return value.

Calling this method is equivalent to setting the Convert property to SVC\_PORT\_TO\_NAME.

The Error property can be checked to obtain the error code returned by the underlying protocol stack.

### Example

```
ServiceName = Socket.SvcPortToName (13, "tcp")  
If ServiceName = "" Then  
    MsgBox "Cannot obtain service name", 64, "Sample Program"  
End If
```

