Fenestra Software L.L.C.

Windows Socket Custom Control Reference

Description

This control provides communications for your application by allowing the transmission and reception of data through the winsock dynamic link library.

File Name

FSSOCKET.VBX

Object Type

FSSocket

Remarks

The FSSocket control has two methods for establishing a connection via winsock.

- **Client mode.** Your application sets the PortNumber and HostAddress it wishes to communicate with, and then uses the Connect property to establish the connection.
- **Server mode.** Your application sets the port it wants to Listen on for a connection. The control will notify the application when a client has made a connection.

Each FSSocket control can communicate only over one connection. If you need more than one connection you must use more than one control.

Distribution Note

When you create and distribute applications that use the FSSocket control, you should install the file FSSOCKET.VBX in the customer's Microsoft Windows \SYSTEM subdirectory. The Setup Kit included with Visual Basic provides tools to help you write setup programs that install your applications correctly.

Application Notes

The following Applications Notes discuss the usage of FSSocket.

- A simple example
- Creating a TCP Client
- Creating a TCP Server
- Creating a UDP Client
- Creating a UDP Server

What's New

- LocalPort Property added.
- After a connection is established, the LocalPort and LocalAddress properties are updated via a call to getsocketname(). This is done to establish the correct LocalAddress for systems with more than one IP address.
- UDP send functionality upgraded. No default address is established, the datagram is sent to the host specified by the HostAddress and PortNumber properties when the Send property is set.
- UDP listen functionality fixed. It works now.

Copyright Notice/License Information

License Agreement

You should carefully read the following terms and conditions before using this software. Unless you have a different license agreement signed by Fenestra Software, your use of this software indicates your acceptance of this license agreement and warranty.

Evaluation and Registration

This is not free software. You are hereby licensed to use this software for evaluation purposes without charge for a period of 60 days. If you use this software after the 60 day evaluation period a registration fee as shown in the section Ordering Information/Order Form is required. Payments must be in US dollars drawn on a US bank, and should be sent to:

Fenestra Software
P.O. Box 87347
Phoenix AZ 85080

When payment is received you will be sent a registered copy of the latest version of FSSocket.

Unregistered use of FSSocket after the 60-day evaluation period is in violation of U.S. and international copyright laws.

Shareware Distribution

Provided that you verify that you are distributing the Shareware Version (double click on the "About" property) you are hereby licensed to make as many copies of the Shareware Version of this software and documentation as you wish; give exact copies of the original Shareware Version to anyone; and distribute the Shareware Version of the software and documentation in its unmodified form via electronic means. There is no charge for any of the above.

You are specifically prohibited from charging, or requesting donations, for any such copies, however made; and from distributing the software and/or documentation with other products (commercial or otherwise) without prior written permission, with one exception:

Disk Vendors approved by the Association of Evaluation Professionals are permitted to redistribute FSSocket, subject to the conditions in this license, without specific written permission.

No one, other that Fenestra Software L.L.C. or its designated representative, may distribute the license files(FSSOCKET.LIC and/or FSSOCKET.SIT).

Registered Version

FSSocket can be used in two modes.

- Design mode, which is used for creating applications.
- Runtime mode, which is used to supply functionality for previously created applications.

Design mode

One registered copy of FSSocket may be used in design mode, by a single person who uses the software personally on one or more computers, or installed on a single workstation used nonsimultaneously by multiple people, but not both.

You may access the registered version of FSSocket through a network, provided that you have obtained individual licenses for the software covering all workstations that will access the software in design mode through the network. For instance, if 8 different workstations will access FSSocket in design mode on the network, each workstation must have its own FSSocket license, regardless of whether they use FSSocket at different times or concurrently.

You may not place the license file on more than one system, or access the license file over a network.

You may not distribute the license files(FSSOCKET.LIC and/or FSSOCKET.SIT).

You may transfer the license file on a permanent basis provide you retain no copies.

Runtime Mode

There are no runtime fees or royalties.

Once you have licensed FSSocket, you have a royalty-free right to reproduce and distribute the FSSOCKET.VBX file as a part of your own products.

You may not distribute the license files (FSSOCKET.LIC and/or FSSOCKET.SIT).

Disclaimer of Warranty

THIS SOFTWARE AND THE ACCOMPANYING FILES ARE SOLD "AS IS" AND WITHOUT WARRANTIES AS TO PERFORMANCE OF MERCHANTABILITY OR ANY OTHER WARRANTIES WHETHER EXPRESSED OR IMPLIED. Because of the various hardware and software environments into which FSSocket may be put, NO WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE IS OFFERED.

Good data processing procedure dictates that any program be thoroughly tested with non-critical data before relying on it. The user must assume the entire risk of using the program. ANY LIABILITY OF THE SELLER WILL BE LIMITED EXCLUSIVELY TO PRODUCT REPLACEMENT OR REFUND OF PURCHASE PRICE.

Ordering Information/Order Form

FSSocket 1.0 Order Form/Invoice

Ordering by check:

To order by check send this order form and a check to:

Fenestra Software L.L.C.
P.O.Box 87347
Phoenix AZ 85080

To print this order form, click on Print Topic in the File pull-down menu. Payments must be in US dollars drawn on a US bank, or you can send international postal money orders in US dollars.

Credit card ordering:

For information on ordering by MasterCard, Visa, American Express, or Discover by phone, FAX, or CompuServe email, or postal mail, click here:Credit Card Ordering Information.

CompuServe Registration:

To have the registration fee added to your CompuServe bill type GO SWREG at the ! prompt and follow the menus.

Purchase Orders:

For information on using purchase orders click here:Purchase Orders.

Order Form

Single Copy Pricing

	Price		Quantity		Amount
	$75	x	_____	=	_____

Multiple Copy Pricing

For 2 to 9 copies	$60	x	_____	=	_____
For 10 to 24 copies	$48	x	_____	=	_____
For 25 to 49 copies	$38	x	_____	=	_____
For 50 to 99 copies	$31	x	_____	=	_____
For 100 to 199 copies	$25	x	_____	=	_____
Shipping					
Arizona residents add 8% sales tax:				+	_____
Total payment:					_____

Name: _____Date:_____
Company: _____
Address: _____
City, State, Zip: _____
Country: _____
Day Phone: _____Eve:_____
Electronic Mail address: _____
How did you hear about FSSocket? _____
Comments:

Credit Card Ordering

You can order with MasterCard, Visa, American Express, or Discover from Public (software) Library by calling 800-2424-PsL or 713-524-6394 or send your order by FAX to 713-524-6398 or by CIS Email to 71355,470 or Internet mail to 71355.470@compuserve.com. You can also mail credit card orders to PsL at P.O.Box 35705, Houston, TX 77235-5705.

Please be sure to include your credit card number and expiration date on all credit card orders.

The above numbers are for credit card orders only.

Fenestra Software cannot be reached at these numbers.

Any questions about the status of the shipment of the order, refunds, registration options, product details, technical support, volume discounts, dealer pricing, site licenses, non-credit card orders, etc, must be directed to Fenestra Software L.L.C., P.O. BOX 87347, Phoenix AZ 85080 or by email to 70176,264 on CompuServe or jcs@primenet.com on Internet.

To insure that you get the latest version, PsL will notify us the day of your order and we will ship the product directly to you.

Purchase Orders

Purchase orders (net 30 days) are accepted only from government and accredited educational institutions and major corporations, provided that they are submitted on purchase order forms with a purchase order number. Please be sure to include the standard FSSocket order form with a purchase order. Due to the extra work involved in processing purchase orders you are encouraged to use a credit card, CompuServe's SWREG registration service, petty cash, or an expense account when possible for small orders.

Support and Questions

Technical support is available at no charge by sending electronic mail to

jcs@primenet.com on the Internet,

or by sending US Mail to

Fenestra Software L.L.C.
P.O.Box 87347
Phoenix AZ 85080

Events

Connected DataReceived Disconnected ReadyToSend

Connected Event

Description

The Connected event is generated when a successful connection has been made.

Visual Basic

Sub *FSSocket*_**Connected (***StatusCode* **As Integer,** *Description* **As String)**

Remarks

The *StatusCode* contains 0 and the *Description* contains "OK".

This event is not generated for client type UDP connections. UDP connections are "send and pray".

DataReceived Event

Description

This event is generated when data has been received over the connection.

Visual Basic

Sub *FSSocket*_**DataReceived (***Text* **As String,** *EOL* **As Integer)**

Remarks

The *Text* contains the data that was received. *EOL* is **True** if the EOL property has been set and a matching string was found in the data received. The matching EOL string is stripped from the *Text*. Any data that remains after the EOL string will be included in the next DataReceived event.

Disconnected Event

Description

The Disconnected event is generated when a connection closes.

Visual Basic

Sub *FSSocket*_**Disconnected (***StatusCode* **As Integer,** *Description* **As String)**

Remarks

The *StatusCode* and *Description* contain the reason for the disconnect.

ReadyToSend Event

Description

The ReadyToSend event is generated when the backlog of data to send reaches 0.

Visual Basic

Sub *FSSocket*_**ReadyToSend ()**

Remarks

If a Send fails with a WSALastErrorMsg of "Operation Would Block", your application can wait for this event before retrying the Send

Properties

About	Connect	EOL	HostAddress
HostName	InputBufferSize	ListenPort	Listen
ListenSocket	LocalAddress	LocalName	LocalPort
OutputBufferSize	PortNumber	Protocol	Send
ServiceName	Socket	WSALastErrorMsg	WSALastError

About Property

Description

Contains Copyright information

Connect Property

Description

Set to **True** to establish a connection.

Remarks

This is an action property. When your application sets this property to **True** the control attempts to create a connection to the PortNumber and HostAddress specified. If the connect is successful the Connected event occurs, otherwise this property is set to **False**.

If the Protocol property is set to UDP, no connection is created. The socket is bound to your local address. To specify a local port number for the bind operation, set ListenPort prior to setting the Connect Property to **True**. If the ListenPort number is zero, winsock will assign a local port number. The ListenPort is used because, if you need to actually set a local port number for a UDP connection, you will probably be "listening" for packets, ie. creating a server.

This property is also set to **True** when the control is in server mode and a connection is established.

EOL Property

Description

Contains a string that delimits lines of input received.

Remarks

When receiving ASCII data it is often useful to break the data up into individual lines. This property allow you to specify a string that marks the end of line for incoming data. The receiving buffer is searched for a matching string. When it is found, the data, up to but not including the end of line string, is sent to your application via the DataReceived event. Any remaining data in the buffer is sent to your application in the next DataReceived event.

HostAddress Property

Description

Contains the Internet Address of the host your application is attempting to communicate with.

Remarks

The format of the property is the standard Internet dot format.

Setting this property causes the control to issue a blocking gethostbyaddr call to fill in the HostName property.

HostName Property

Description

Contains the Internet Domain Name of the host your application is attempting to communicate with.

Remarks

The format of the property is the standard Internet dot format.

Setting this property causes the control to issue a blocking gethostbyname call to fill in the HostAddress property.

InputBufferSize Property

Description

Specifies the size of the input buffer.

Remarks

This specifies the size of the buffer FSSocket uses to hold the incoming data prior to generating the DataReceived event and the size of the underlying Window Socket input buffer.

Listen Property

Description

Set to **True** to establish a server application.

Remarks

This is an action property. Setting this property to True puts the control into server mode. The control will then listen on the specified port for a client connection. When a connection is made a Connected event occurs. At this time your application can examine the HostAddress property to establish the address of the client. While a client/server connection is open any other attempt by a client to open a connection will be queued until the current connection is closed.

When your application sets this property to **True** the control attempts to listen on the ListenPort specified.

If the ListenPort is zero, Sockets will select a port for you and the port number will be placed in ListenPort.

If the Protocol property is set to UDP, FSSocket will watch the ListenPort for UDP datagrams. When one arrives the DataReceived event is fired.

ListenPort Property

Description

Specifies the port that your application wishes to monitor for incoming connections.

Remarks

Setting this property to zero causes Sockets to select a port number when the Listen property is set to True.

ListenSocket Property

Description

Contains the Socket number this control is watching for connections.

Remarks

This property is read only.

LocalAddress Property

Description

Contains the Internet Address of the system this control is running on.

Remarks

This property is read only.

LocalName Property

Description

Contains the Domain Name of the system this control is running on.

Remarks

This property is read only.

LocalPort Property

Description

Contains the local port number for the current connection.

Remarks

This property is read only.

OutputBufferSize Property

Description

Specifies the size of the output buffer.

Remarks

This specifies the size of the buffer FSSocket uses to hold the outgoing data while waiting for a FD_WRITE message from Windows Sockets and the size of the underlying Window Socket output buffer.

PortNumber Property

Description

Specifies the number of the port your application is attempting to communicate with.

Remarks

Setting this value also sets the ServiceName property.

Protocol Property

Description

Specifies the protocol to use for this connection.

Remarks

Currently only TCP and UDP are supported.

Connected events are not generated when using UDP for a client connections.

Send Property

Description

Send data over the connection.

Remarks

Assigning a string to this property sends the string over the connection.

ServiceName Property

Description

Contains a string representing the service you wish to communicate with.

Remarks

Setting this value also sets the PortNumber property.

Socket Property

Description

Contains the Socket number for this control.

Remarks

This property is read only.

WSALastError Property

Description

Contains the last error number returned by Windows Sockets.

Remarks

When FSSocket detects an error from Windows Sockets, the error number is placed in this property and a trappable FSSocket error is generated.

WSALastErrorMsg Property

Description

Contains a string corresponding to the WSALastError property.

Remarks

When FSSocket detects an error from Windows Sockets, an error message is placed in this property and a trappable FSSocket error is generated.

Error Codes

The following trappable errors can be generated by the FSSocket control.

Error Number	Explanation
20101	Unable to obtain memory.
20102	WinSocket Error detected.

The following is a list of possible error codes returned in the WSALastError property, along with their explanations. The error numbers are consistently set across all Windows Sockets compliant implementations.

WSALastError	WSALastErrorMsg
10004	Interrupted system call.
10009	Bad file number.
10013	Access denied.
10014	Bad address.
10022	Invalid argument.
10024	Too many open files.
10035	Operation would block.
10036	Operation now in progress.
10037	Operation already in progress.
10038	Socket operation on non-socket.
10039	Destination address required.
10040	Message too long.
10041	Protocol is wrong type for socket.
10042	Bad protocol option.
10043	Protocol not supported.
10044	Socket type not supported.
10045	Operation not supported on socket.
10046	Protocol family not supported.
10047	Address family not supported by protocol family.
10048	Address already in use.
10049	Cant assign requested address.
10050	Network is down.
10051	ICMP network unreachable.
10052	Network was reset.
10053	Software caused connection abort.
10054	Connection reset by peer.
10055	No buffer space is supported.
10056	Socket is already connected.
10057	Socket is not connected.
10058	Cant send after socket shutdown.
10059	Too many references.
10060	Connection timed out.
10061	Connection refused.
10062	Too many levels of symbolic links.
10063	Name too long.
10064	Host is down.
10065	Host is unreachable.
10066	Directory not empty.
10067	EPROCLIM returned.

10068	EUSERS returned.
10069	Disk quota exceeded.
10070	ESTALE returned.
10071	The object is remote.
10091	System not ready.
10092	Version is not supported.
10093	Not initialized.
10101	The circuit has gracefully terminated.
11001	Host not found.
11002	Try again.
11003	Non recoverable error.
11004	No data record available.

A simple example

Probably the simplest example program that can be written is a hostname lookup.

1. Start Visual Basic.
2. From the file menu, use Add File to add FSSOCKET.VBX to the project.
3. Place the following on the form.
   A text box named HostName.
   A text box named HostAddress.
   A command button named getname.
   A command button named getaddr.
   A fssocket control named fssocket1.
4. Add the following code:

```
Sub getaddr_Click ()
    fssocket1.HostName = hostname.Text
    HostAddress.Text = fssocket1.HostAddress
End Sub

Sub getname_Click ()
    fssocket1.HostAddress = HostAddress.Text
    HostName.Text = fssocket1.HostName
End Sub
```

5. Run the program.

   Type a domain name into the HostName text box and click on the getaddr button. The IP address that corresponds to the domain name will be placed in the HostAddress text box.

   Type an IP address in the HostAddress box and click on the getname button. The domain name for that IP address will be placed in the HostName box.

Creating a TCP Client

The TCP protocol is meant for reliable communications over a TCP/IP network. Reliable means that the data is guaranteed to arrive complete and in the correct order in which it was sent.

A simple example

For our example we will connect to the echo port on the local host machine. This port simply echoes back anything it receives.

1. Start Visual Basic.
2. From the file menu, use Add File to add FSSOCKET.VBX to the project.
3. Place the following on the form and set the properties as shown:
   A text box named IBox.
   A text box named Obox.
   A fssocket control named fssocket1.
   Set HostName to "localhost" or any other host that has an ECHO server.
   Set Service to "echo" or Set PortNumber to 7.
4. Add the following code:

```
Sub Form_Load ()
    fssocket1.Connect = True
End Sub
Sub IBox_KeyPress (keyascii As Integer)
    If keyascii = 13 Then
        fssocket1.Send = IBox.Text
        keyascii = 0
        IBox.Text = ""
    End If
End Sub
Sub FSSocket1_DataReceived (Text As String, EOL As Integer)
    OBox.Text = Text
End Sub
```

5. Run the program.

   Type some text into IBox and then hit enter. The contents of IBox are sent to the echo port on the specified host. The data is echoed back and placed in OBox when it is received.

   This program doesnt do anything real useful other than demonstrate that FSSocket is working.

Programming considerations

Most client/server network communications protocols require a "conversational" mode of interaction. The client sends a request and waits for the server to respond. Unfortunately windows being message based, does not lend itself to this type of interaction.

One way of over coming this, is to make your DataReceived function into a "state" machine that knows how to handle any possible response from the server.

Another way involves setting and testing some global variables, for example:

```
Dim WaitingForData as Integer
Dim ServerData as String
Sub FSSocket1_DataReceived (Text As String, EOL As Integer)
    ServerData = Text
    WaitingForData = False
End Sub
```

In your main line code:

```
WaitingForData = True
fssocket1.send = "Command to Server"
```

```
While WaitingForData = True
    DoEvents
Wend
```
At this point the global ServerData contains the response from the server.

Creating a TCP Server

A simple example

For our example we will create a simple echo server. This server simply echoes back anything it receives.

1. Start Visual Basic.
2. From the file menu, use Add File to add FSSOCKET.VBX to the project.
3. Place the following on the form and set the properties as shown:

   A text box named Portnum.

   A fssocket control named fssocket1.
4. Add the following code:

```
Sub Form_Load ()
    FSSocket1.ListenPort = 0
    FSSocket1.Listen = True
    portnum = FSSocket1.ListenPort
End Sub
Sub FSSocket1_DataReceived (text As String, EOL As Integer)
    FSSocket1.Send = text
End Sub
Sub FSSocket1_Connected (StatusCode As Integer, Description As String)
    Debug.Print "Connected to " + FSSocket1.HostAddress + " on port " +
FSSocket1.LocalPort
End Sub
```

5. Run the program.

   The number that is shown in the portnum text box is the local port number that winsock assigned. To assign a specific port, set the ListenPort property prior to setting Listen to True.

Creating a UDP Client

UDP is a conectionless protocol. This means that there is no end to end connection between client and server. Each transmission is simply sent. There is no guarantee that the data arrived or in what order the data was received. If two datagrams are sent, the second datagram may be received before the first.

A simple example

For our example we will connect to the echo port on the local host machine. This port simply echoes back anything it receives.

1. Start Visual Basic.
2. From the file menu, use Add File to add FSSOCKET.VBX to the project.
3. Place the following on the form and set the properties as shown:
   A text box named IBox.
   A text box named Obox.
   A fssocket control named fssocket1.
   Set Protocol to UDP.
4. Add the following code:

```
Sub Form_Load ()
    fssocket1.Connect = True
End Sub
Sub IBox_KeyPress (keyascii As Integer)
    If keyascii = 13 Then
        fssocket1.PortNumber = 7
        fssocket1.HostAddress = "127.0.0.1"
        fssocket1.Send = IBox.Text
        keyascii = 0
        IBox.Text = ""
    End If
End Sub
Sub FSSocket1_DataReceived (Text As String, EOL As Integer)
    OBox.Text = Text
End Sub
```

5. Run the program.

   Type some text into IBox and then hit enter. The contents of IBox are sent to the echo port on the specified host. The data is echoed back and placed in OBox when it is received.

   As you can see this is almost identical to the TCP client example, but what happens in FSSocket is quite different.

   The first difference is that no HostAddress or PortNumber were specified before the Connect. This is because the UDP connect only binds our local address. We do not need to set a destination until we are ready to send data. If you are sending to only one address/port, you should only set these values once, because each time a HostAddress or HostName is set the nameservers are queried. This can slow down performance.

   The second difference is that the Connect and Disconnect events never fire.

Creating a UDP Server

A simple example

For our example we will create a simple echo server. This server simply echoes back anything it receives.

1. Start Visual Basic.
2. From the file menu, use Add File to add FSSOCKET.VBX to the project.
3. Place the following on the form and set the properties as shown:

   A text box named Portnum.

   A fssocket control named fssocket1.

   Set Protocol to UDP.
4. Add the following code:

```
Sub Form_Load ()
    FSSocket1.ListenPort = 0
    FSSocket1.Listen = True
    portnum = FSSocket1.ListenPort
End Sub
Sub FSSocket1_DataReceived (text As String, EOL As Integer)
    Debug.Print "Data received from  " + FSSocket1.HostAddress + " port "
+ FSSocket1.PortNumber
    FSSocket1.Send = text
End Sub
```

5. Run the program.

   The number that is shown in the portnum text box is the local port number that winsock assigned. To assign a specific port, set the ListenPort property prior to setting Listen to True.

   As you can see this is almost identical to the TCP server example, but what happens in FSSocket is quite different.

   The first difference is that HostAddress or PortNumber properties are updated by FSSocket each time data is received. This is because the UDP each datagram can come from a different client.

   The second difference is that the Connect and Disconnect events never fire.

   The third difference is that the EOL property is ignored. Since each datagram can come from a different client, FSSocket cannot attempt to break up a datagram.