Chat Uygulaması

Ekteki doküman Microsoft.com sitesinden alınmış bir örnektir. VB 5.0 kullanıcıları bu dökümana VB Boks Online seçeneğinden ulaşabilir.

Sunucu Uygulaması

Projenize bir form ekleyiniz aşağıdaki denetimleri form üzerine yerleştiriniz. Denetim adları (name) bitişik açıklamalarda verilmiştir.

İstemci Uygulaması

Projenize yeni bir form ekleyiniz ve aşağıdaki denetimleri ekleyiniz.
Sunucu Kaynak Kodları

Sub Form_Load()
    TCPServer.Protocol = sckTCPProtocol
    TCPServer.LocalPort = 55555
    TCPServer.Listen
    frmClient.Show
End Sub

Sub tcpServer_ConnectionRequest(ByVal requestID As Long)
    If TCPServer.State <> sckClosed Then TCPServer.Close
    TCPServer.Accept requestID
End Sub

Sub txtSendData_Change()
    TCPServer.SendData txtSendData.Text
End Sub

Sub tcpServer_DataArrival(ByVal bytesTotal As Long)
    Dim strData As String
    TCPServer.GetData strData
    txtOutput.Text = strData
End Sub

İstemci Kaynak Kodları

Sub Form_Load()
    tcpClient.Protocol = sckTCPProtocol
    tcpClient.RemoteHost = "127.0.0.1"
    tcpClient.RemotePort = 55555
End Sub

Sub cmdConnect_Click()
    tcpClient.Connect
End Sub

Sub txtSendData_Change()
    tcpClient.SendData txtSendData.Text
End Sub

Sub tcpClient_DataArrival(ByVal bytesTotal As Long)
    Dim strData As String
    tcpClient.GetData strData
    txtOutput.Text = strData
End Sub
A WinSock control allows you to connect to a remote machine and exchange data using either the User Datagram Protocol (UDP) or the Transmission Control Protocol (TCP). Both protocols can be used to create client and server applications. Like the Timer control, the WinSock control doesn't have a visible interface at run time.

**Possible Uses**

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

**Selecting a Protocol**

When using the WinSock control, the first consideration is whether to use the TCP or the UDP protocol. The major difference between the two lies in their connection state:

- The TCP protocol control is a connection-based protocol, and is analogous to a telephone — the user must establish a connection before proceeding.
- The UDP protocol is a connectionless protocol, and the transaction between two computers is like passing a note: a message is sent from one computer to another, but there is no explicit connection between the two. Additionally, the maximum data size of individual sends is determined by the network.

The nature of the application you are creating will generally determine which protocol you select. Here are a few questions that may help you select the appropriate protocol:

1. Will the application require acknowledgment from the server or client when data is sent or received? If so, the TCP protocol requires an explicit connection before sending or receiving data.
2. Will the data be extremely large (such as image or sound files)? Once a connection has been made, the TCP protocol maintains the connection and ensures the integrity of the data. This connection, however, uses more computing resources, making it more "expensive."
3. Will the data be sent intermittently, or in one session? For example, if you are creating an application that notifies specific computers when certain tasks have completed, the UDP protocol may be more appropriate. The UDP protocol is also more suited for sending small amounts of data.

**Setting the Protocol**

To set the protocol that your application will use: at design-time, on the Properties window, click Protocol and select either sckTCPProtocol, or sckUDPProtocol. You can also set the Protocol property in code, as shown below:

```plaintext
Winsock1.Protocol = sckTCPProtocol
```

**Determining the Name of Your Computer**

To connect to a remote computer, you must know either its IP address or its “friendly name.” The IP address is a series of three digit numbers separated by periods (xxx.xxx.xxx.xxx). In general, it's much easier to remember the friendly name of a computer.

**To find your computer's name**

1. On the Taskbar of your computer, click Start.
2. On the Settings item, click the Control Panel.
3. Double-click the Network icon.
4. Click the Identification tab.
5. The name of your computer will be found in the Computer name box.

Once you have found your computer's name, it can be used as a value for the RemoteHost property.
TCP Connection Basics

When creating an application that uses the TCP protocol, you must first decide if your application will be a server or a client. Creating a server means that your application will "listen," on a designated port. When the client makes a connection request, the server can then accept the request and thereby complete the connection. Once the connection is complete, the client and server can freely communicate with each other.

The following steps create a rudimentary server:

To create a TCP server
1. Create a new Standard EXE project.
2. Change the name of the default form to frmServer.
3. Change the caption of the form to "TCP Server."
4. Draw a Winsock control on the form and change its name to tcpServer.
5. Add two TextBox controls to the form. Name the first txtSendData, and the second txtOutput.
6. Add the code below to the form.

```vbnet
Private Sub Form_Load()
    ' Set the LocalPort property to an integer.
    ' Then invoke the Listen method.
    tcpServer.LocalPort = 1001
    tcpServer.Listen
    frmClient.Show ' Show the client form.
End Sub

Private Sub tcpServer_ConnectionRequest ByVal requestID As Long
    ' Check if the control's State is closed. If not,
    ' close the connection before accepting the new
    ' connection.
    If tcpServer.State <> sckClosed Then
        tcpServer.Close
    ' Accept the request with the requestID
    ' parameter.
    tcpServer.Accept requestID
End Sub

Private Sub txtSendData_Change()
    ' The TextBox control named txtSendData
    ' contains the data to be sent. Whenever the user
    ' types into the textbox, the string is sent
    ' using the SendData method.
    tcpServer.SendData txtSendData.Text
End Sub

Private Sub tcpServer_DataArrival ByVal bytesTotal As Long
    ' Declare a variable for the incoming data.
    ' Invoke the GetData method and set the Text
    ' property of a TextBox named txtOutput to
    ' the data.
    Dim strData As String
    tcpServer.GetData strData
txtOutput.Text = strData
End Sub
```

The procedures above create a simple server application. However, to complete the scenario, you must also create a client application.

To create a TCP client
1. Add a new form to the project, and name it frmClient.
2. Change the caption of the form to TCP Client.
3. Add a Winsock control to the form and name it tcpClient.
4 Add two TextBox controls to frmClient. Name the first txtSend, and the second txtOutput.
5 Draw a CommandButton control on the form and name it cmdConnect.
6 Change the caption of the CommandButton control to Connect.
7 Add the code below to the form.

Important Be sure to change the value of the RemoteHost property to the friendly name of your computer.

```vba
Private Sub Form_Load()
    ' The name of the Winsock control is tcpClient.
    ' Note: to specify a remote host, you can use
    ' either the IP address (ex: "121.111.1.1") or
    ' the computer's "friendly" name, as shown here.
    tcpClient.RemoteHost = "RemoteComputerName"
    tcpClient.RemotePort = 1001
End Sub

Private Sub cmdConnect_Click()
    ' Invoke the Connect method to initiate a
    ' connection.
    tcpClient.Connect
End Sub

Private Sub txtSendData_Change()
    tcpClient.SendData txtSend.Text
End Sub

Private Sub tcpClient_DataArrival(ByVal bytesTotal As Long)
    Dim strData As String
    tcpClient.GetData strData
    txtOutput.Text = strData
End Sub
```

The code above creates a simple client-server application. To try the two together, run the project, and click Connect. Then type text into the txtSendData TextBox on either form, and the same text will appear in the txtOutput TextBox on the other form.

Accepting More than One Connection Request

The basic server outlined above accepts only one connection request. However, it is possible to accept several connection requests using the same control by creating a control array. In that case, you do not need to close the connection, but simply create a new instance of the control (by setting its Index property), and invoking the Accept method on the new instance.

The code below assumes there is a Winsock control on a form named sckServer, and that its Index property has been set to 0; thus the control is part of a control array. In the Declarations section, a module-level variable intMax is declared. In the form's Load event, intMax is set to 0, and the LocalPort property for the first control in the array is set to 1001. Then the Listen method is invoked on the control, making it the "listening control. As each connection request arrives, the code tests to see if the Index is 0 (the value of the "listening" control). If so, the listening control increments intMax, and uses that number to create a new control instance. The new control instance is then used to accept the connection request.

```vba
Private intMax As Long

Private Sub Form_Load()
    intMax = 0
    sckServer(0).LocalPort = 1001
    sckServer(0).Listen
End Sub

Private Sub sckServer_ConnectionRequest(Index As Integer, ByVal requestID As Long)
    If Index = 0 Then
        intMax = intMax + 1
```

The code above creates a simple client-server application. To try the two together, run the project, and click Connect. Then type text into the txtSendData TextBox on either form, and the same text will appear in the txtOutput TextBox on the other form.
Load sckServer(intMax)
sckServer(intMax).LocalPort = 0
sckServer(intMax).Accept requestID
Load txtData(intMax)
End If
End Sub

**UDP Basics**

Creating a UDP application is even simpler than creating a TCP application because the UDP protocol doesn't require an explicit connection. In the TCP application above, one Winsock control must explicitly be set to "listen," while the other must initiate a connection with the Connect method.

In contrast, the UDP protocol doesn't require an explicit connection. To send data between two controls, three steps must be completed (on both sides of the connection):

1. Set the RemoteHost property to the name of the other computer.
2. Set the RemotePort property to the LocalPort property of the second control.
3. Invoke the Bind method specifying the LocalPort to be used. (This method is discussed in greater detail below.)

Because both computers can be considered "equal" in the relationship, it could be called a peer-to-peer application. To demonstrate this, the code below creates a "chat" application that allows two people to "talk" in real time to each other:

▶ **To create a UDP Peer**

1. Create a new Standard EXE project.
2. Change the name of the default form to frmPeerA.
3. Change the caption of the form to "Peer A."
4. Draw a Winsock control on the form and name it udpPeerA.
5. On the Properties page, click **Protocol** and change the protocol to UDPProtocol.
6. Add two TextBox controls to the form. Name the first txtSend, and the second txtOutput.
7. Add the code below to the form.

```vbs
Private Sub Form_Load()
    ' The control's name is udpPeerA
    With udpPeerA
        ' IMPORTANT: be sure to change the RemoteHost value to the name of your computer.
        .RemoteHost = "PeerB"
        .RemotePort = 1001 ' Port to connect to.
        .Bind 1002 ' Bind to the local port.
    End With
    frmPeerB.Show ' Show the second form.
End Sub

Private Sub txtSend_Change()
    ' Send text as soon as it's typed.
    udpPeerA.SendData txtSend.Text
End Sub

Private Sub udpPeerA_DataArrival _
    (ByVal bytesTotal As Long)
    Dim strData As String
    udpPeerA.GetData strData
    txtOutput.Text = strData
End Sub
```

▶ **To create a second UDP Peer**

1. Add a standard form to the project.
2. Change the name of the form to frmPeerB.
3. Change the caption of the form to "Peer B."
4 Draw a Winsock control on the form and name it udpPeerB.
5 On the **Properties** page, click **Protocol** and change the protocol to UDPProtocol.
6 Add two TextBox controls to the form. Name the TextBox txtSend, and the second txtOutput.
7 Add the code below to the form.

```vbscript
Private Sub Form_Load()
    ' The control's name is udpPeerB.
    With udpPeerB
        ' IMPORTANT: be sure to change the RemoteHost
        ' value to the name of your computer.
        .RemoteHost = "PeerA"
        .RemotePort = 1002 ' Port to connect to.
        .Bind 1001 ' Bind to the local port.
    End With
End Sub

Private Sub txtSend_Change()
    ' Send text as soon as it's typed.
    udpPeerB.SendData txtSend.Text
End Sub

Private Sub udpPeerB_DataArrival(ByVal bytesTotal As Long)
    Dim strData As String
    udpPeerB.GetData strData
    txtOutput.Text = strData
End Sub
```

To try the example, press F5 to run the project, and type into the txtSend TextBox on either form. The text you type will appear in the txtOutput TextBox on the other form.

**About the Bind Method**

As shown in the code above, you must invoke the Bind method when creating a UDP application. The Bind method "reserves" a local port for use by the control. For example, when you bind the control to port number 1001, no other application can use that port to "listen" on. This may come in useful if you wish to prevent another application from using that port.

The Bind method also features an optional second argument. If there is more than one network adapter present on the machine, the **LocalIP** argument allows you to specify which adapter to use. If you omit the argument, the control uses the first network adapter listed in the Network control panel dialog box of the computer's Control Panel Settings.

When using the UDP protocol, you can freely switch the RemoteHost and RemotePort properties while remaining bound to the same LocalPort. However, with the TCP protocol, you must close the connection before changing the RemoteHost and RemotePort properties.