

Week 5 – ActiveX and Winsock

ActiveX – What’s all the hype?

- Distributed computing (integrate personal machine to WWW)
- It’s a (distributed computing) technology
 - Controls
 - Components
 - Documents
- Reusable binary code (component) developed under any language capable of generating ActiveX binaries
- Modular and interchangeable

ActiveX and acronyms:

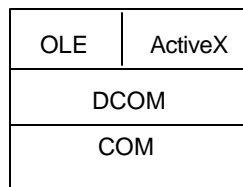


Figure 1: ActiveX technologies

Component Object Model (COM)

- Heart of reusable component development; language/location/compiler independent
- Binary-level specification describing how components communicate (interoperability)
- Make applications more dynamic, customizable and flexible
- It’s the foundation for ActiveX

Distributed COM (DCOM)

- COM “with a longer wire”; execute COM components remotely
- Collectively solve complex problems using remote machines
- Security
- Enterprise level building block

Object Linking and Embedding (OLE)

- Technology allowing applications to work together (e.g. Excel and Word) “containers”
- Services: drag, drop and automation (one app controls another)

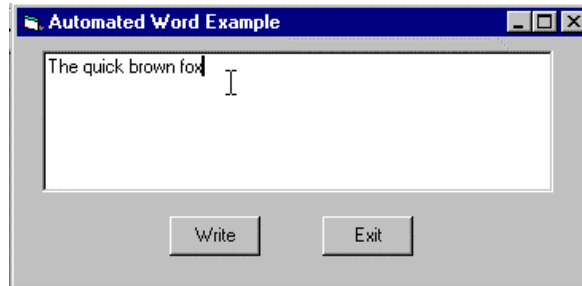
ActiveX

- Reusable components that can be “plugged” into VB, VC++ etc
- Examples: calendars, spell checker, image editor, Winsock
- Natural evolution from OLE
- VB 5.0 and 6.0 can create ActiveX controls
- VBX (VB extensions) cannot be used in VB 5.0 and 6.0

Example: ActiveX (rather than OLE) automation using Word

Objective: Clicking the write command button will save whatever is in the text box, into a Word DOC file called myWord.doc

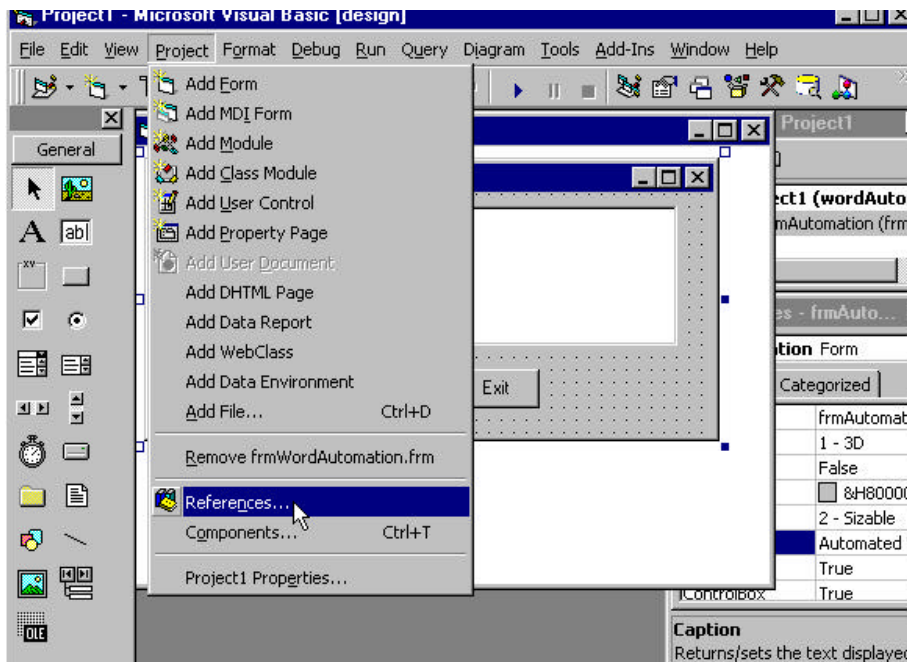
Step 1: Create the GUI



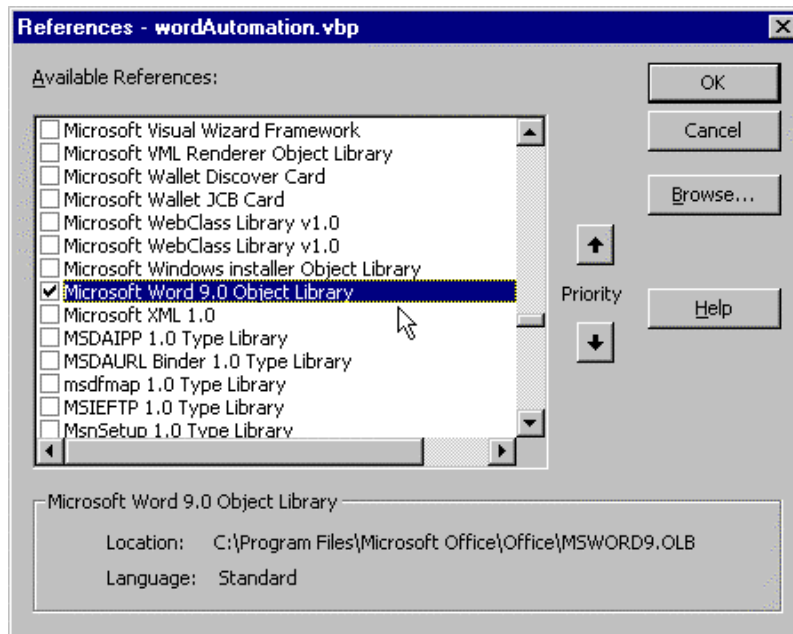
Project: wordAutomation
 Form: frmWordAutomation
 Form: Name: frmAutomation
 Caption: Automation Word Example
 Text box: Name: txtDisplay
 Command Name: cmdWrite
 Command Name: cmdExit

Step 2: Microsoft Word ActiveX component (NB: assumes Microsoft Office installed previously)

Select **Project-References**



Scroll down, look for and check off **“Microsoft Word 9.0 Object Library”**, click OK



Step 3: Code Programming step

```
'FILE: frmAutomation
'Demonstrates automation using Word
Option Explicit
Dim mWord As Word.Application

Private Sub cmdExit_Click()
    End
End Sub

Private Sub cmdWrite_Click()
    cmdWrite.Enabled = False
    Call mWord.Documents.Add
    mWord.Selection.Shading.Texture = wdTexture22Pt5Percent
    mWord.Selection.Font.Size = 30
    Call mWord.Selection.TypeText(txtInput.Text)
    Call
mWord.Documents(1).SaveAs("C:\00Courses\mem800\week05\vbPrograms\myWord.doc")
    Call mWord.Quit
End Sub

Private Sub Form_Initialize()
    Set mWord = New Word.Application
End Sub

Private Sub Class_Terminate()
    Set mWord = Nothing
End Sub
```

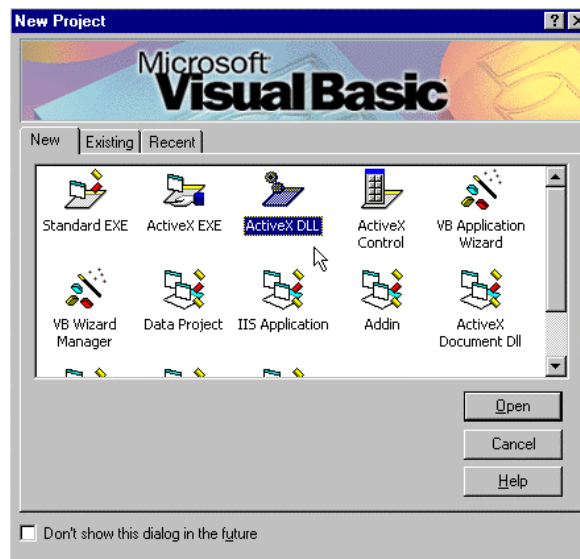
Explore:

- Open myWord.doc – it has whatever was typed in txtInput.text
Note the font size, and shading (Format-Borders and Shading-Shading) value
- Change font size: mWord.SelectionFont.Size = 12
- Change shading: mWord.Selection.Shading.Texture = wdTexture40Percent

Example: ActiveX DLL

- VB allows one to create DLL's; it isn't executable but is to be used by other programs

Objective: Word's built-in spellchecker can be recycled for our own stand-alone program

Step 1: Start an ActiveX DLL project

Save Class as classActiveXDll
Save Project as projectActiveXDll

Step 2: Code programming

```
'FILE: classActiveXDll
'ActiveX DLL example
Option Explicit
Private mWordRef As Word.Application

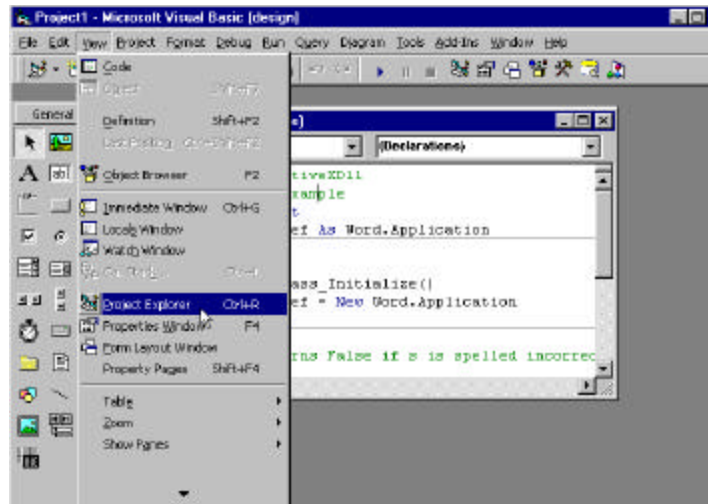
Private Sub Class_Initialize()
    Set mWordRef = New Word.Application
End Sub

'Function returns False if s is spelled incorrectly
Public Function SpellCheckWord(ByVal s As String) As Boolean
    SpellCheckWord = mWordRef.CheckSpelling(s)
End Function

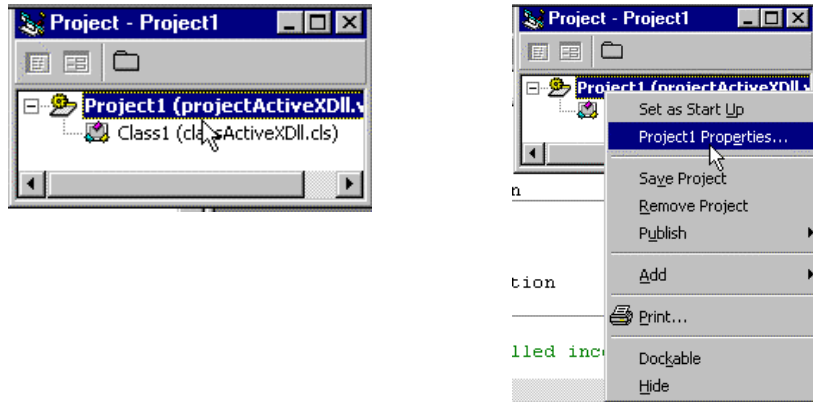
Private Sub Class_Terminate()
    Call mWordRef.Quit
    Set mWordRef = Nothing
End Sub
```

Step 3: Give your DLL an appropriate description

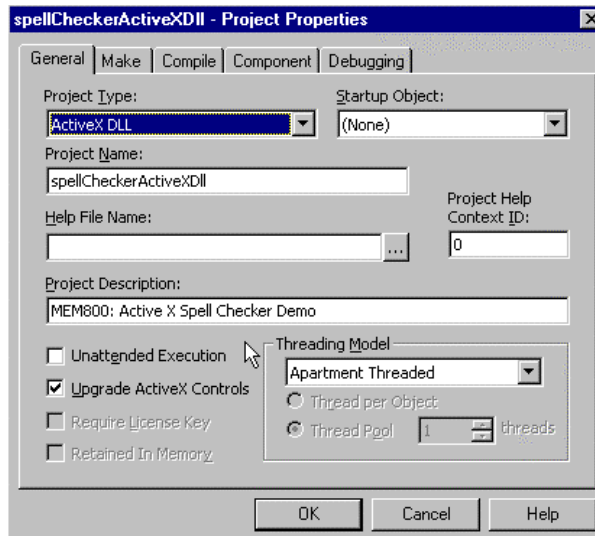
View – Project Explorer



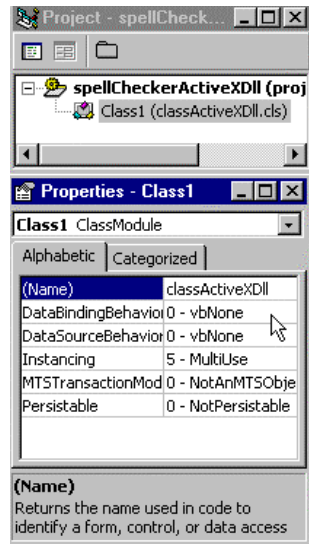
Click once on **Project1** to turn blue, right click and select **Project Properties**



Change **Project Name** and add **Project Description**



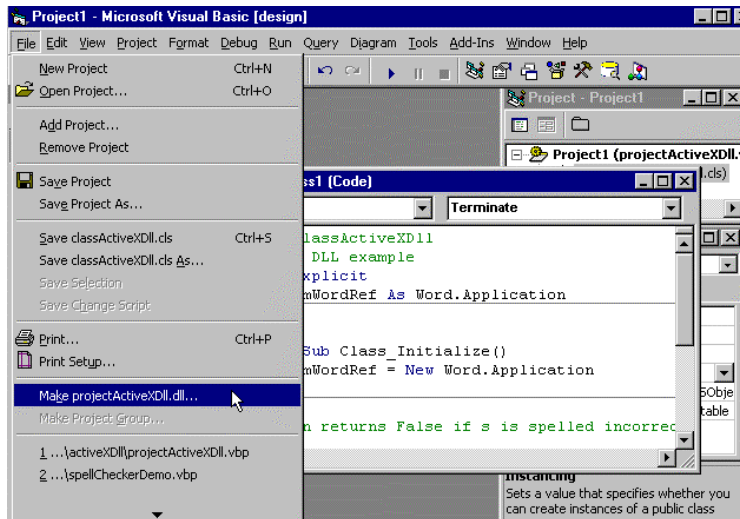
Step 4: Name your class – click on Class1 and change name to: **classActiveXDll**



Step 5: Compile

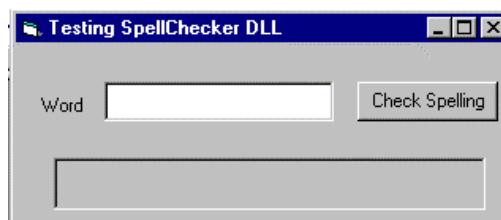
First make sure that from **Project-References**, “**Microsoft Word 9.0 Object Library**” is checked off.

Save, then **Files-Make projectActiveXDll.dll**



Step 6: Create a program (EXE file) that uses projectActiveXDll.dll

GUI:

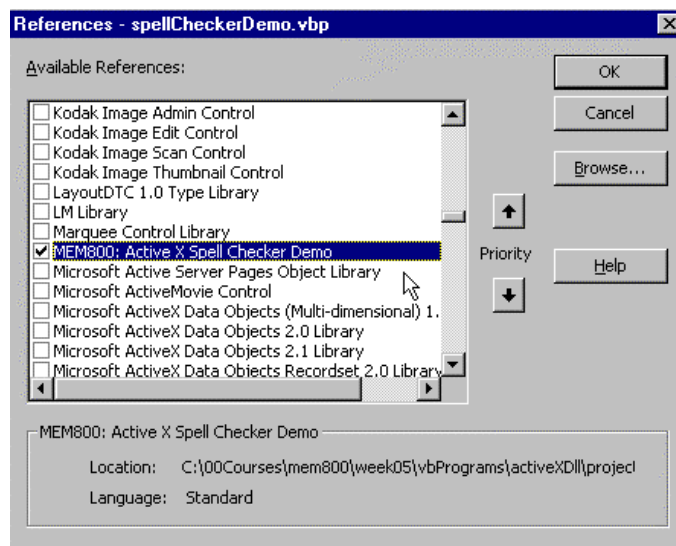


Save Project as: spellCheckerDemo
 Save Form as: spellCheckDemo

Form	Name:	frmSpellCheckDemo
	Caption:	Testing SpellChecker DLL
Command	Name:	cmdCheck
Label	Name:	lblTextInput
	Caption:	Word:
Text box	Name:	txtInput
Label:	Name:	lblDisplay
	BorderStyle:	1 – Fixed Single

Step 7: Include the DLL you created in Step 3 (MEM800: ActiveX Spell Checker Demo)

View – Resources



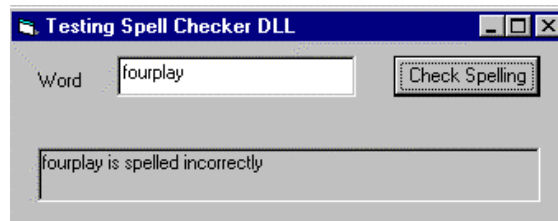
Also make sure that **“Microsoft Word 9.0 Object Library”** is also checked

Step 8: Code Programming

```
'FILE: frmSpellCheckDemo
'Form module to exercise SpellChecker DLL
Option Explicit
Private mChecker As classActiveXDll ` recall name of class
```

```
Private Sub cmdCheck_Click()  
    If mChecker.SpellCheckWord(txtInput.Text) Then  
        lblDisplay.Caption = txtInput.Text & " is spelled correctly"  
    Else  
        lblDisplay.Caption = txtInput.Text & " is spelled incorrectly"  
    End If  
End Sub  
  
Private Sub Form_Initialize()  
    Set mChecker = New classActiveXDll ` recall name of class  
End Sub  
  
Private Sub Form_Terminate()  
    Set mChecker = Nothing  
End Sub
```

After saving everything, run your program and test:



Example: Off-the-shelf ActiveX Components

- ActiveX is a technology – components can be created in VB, VC++ and other compilers
- Why re-invent the wheel?
- 3rd party vendors supply ActiveX components e.g. LabView

Objective: Utilize a shareware LCD ActiveX component (called axpanell.zip – axpanel.dll)

Sample Sources:

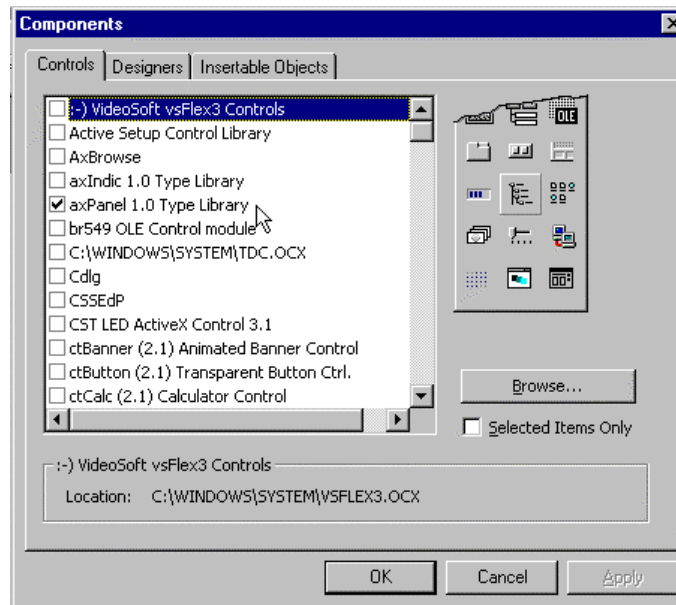
1. VB Zone: <http://pdhut.www5.50megs.com/vbzone/controls/listindex.htm>
2. Download 32: <http://www.download32.com/windows/Development/ActiveX/default.htm>
3. Winsite <http://www.winsite.com/webdev/activex/>
4. Code Guru: <http://www.vbextras.com/categories/gauge+dial+metercomponent.asp>
5. Total Shareware: http://www.totalshareware.com/ASP/list_view.asp?catid=10&Page=4
6. Yuris Puteinis LCD panel ActiveX: <http://www.geocities.com/SiliconValley/Lab/4340/>
Download: axPanel.zip (contains ActiveX component) and panelVb.zip (VB examples)

Step 1: With ActiveX component in hand, need to register it. Oftentimes shareware comes with a registration program e.g. RegAX

Alternatively: Start-Run regsvr32 drive:\path\filename

To un-register: Start-Run regsvr32 /u drive:\path\filename

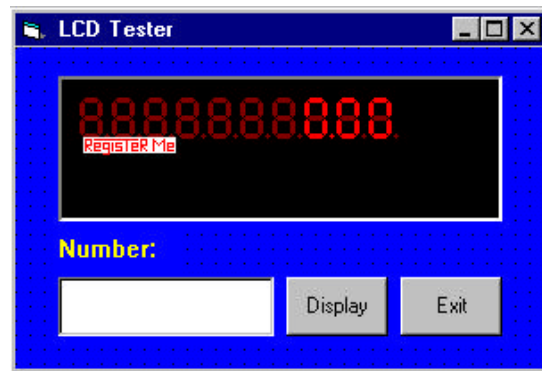
Step 2: Make sure Projects-Register has axPanel 1.0 Type Library checked
Make sure Projects-Components has axPanel 1.0 Type Library checked



This will result in having the axPanel ActiveX component show on your controls toolbox:



Step 3: Construct your GUI:



Step 4: Code programming

```
Private Sub cmdDisplay_Click()  
    myAxPanel.Text = txtInput.Text  
End Sub
```

That's it! Oftentimes distributed ActiveX components come with explanations of code programming use.

Winsock

What is Winsock?

- Enables client/server application programming using either TCP (transmission control protocol) or UDP (user datagram protocol)
- It operates on the lowest level of all Internet controls e.g. WebBrowser, Internet Transfer (hypertext transfer protocol and file transfer protocol)

TCP

- Similar to telephone conversation (connection is needed before conversation)
- Use for transmitting large amounts of data and when reliability imperative

UDP

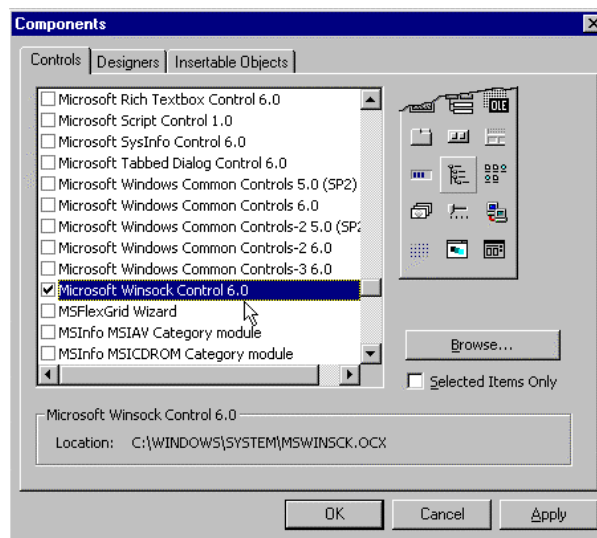
- Like mail (packets are created, addressed and sent over network)
- Small data sent intermittently

Best of all: Winsock can be treated like any other VB component

Example: Client-Server application using Winsock

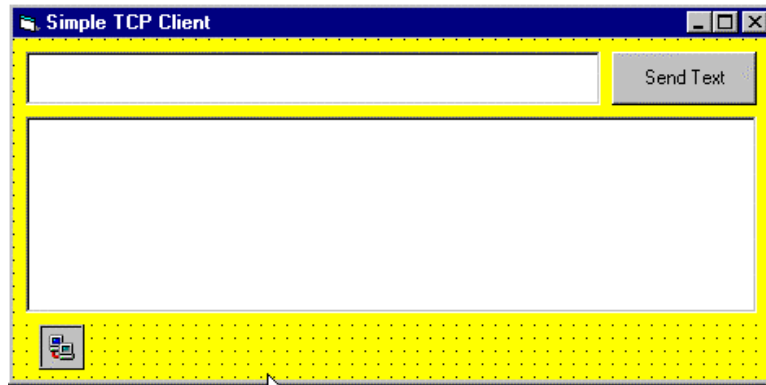
Objective: Create a “chat” program

Step 1: Create the Client with Project and Form names tcpClientDemo and frmTcpClientDemo respectively. Enable Winsock component using Project-Components: **Microsoft Winsock Control 6.0**



Note the Winsock control will be at bottom of toolbox:



Step 2: Setup Client GUI**Step 3: Client code programming**

```
' FILE: tcpClient
' DATE: 01/15/00 12:45
' AUTH: P.Oh
' DESC: A Simple TCP Client
' REFS: Deitel p. 829 Fig. 19.12
Option Explicit

Private Sub Form_Load()

    cmdSend.Enabled = False

    ' Set up local port and wait for connection
    tcpClient.RemoteHost = InputBox("Enter the remote host IP Address", _
        "IP Address", "localhost")

    If tcpClient.RemoteHost = "" Then
        tcpClient.RemoteHost = "localhost"
    End If

    tcpClient.RemotePort = 5000 ' server port
    Call tcpClient.Connect ' connect to RemoteHost address

End Sub

Private Sub Form_Resize()
    On Error Resume Next
    Call cmdSend.Move(ScaleWidth - cmdSend.Width, 0)
    Call txtSend.Move(0, 0, ScaleWidth - cmdSend.Width)
    Call txtOutput.Move(0, txtSend.Height, ScaleWidth, _
        ScaleHeight - txtSend.Height)
End Sub

Private Sub Form_Terminate()
    Call tcpClient.Close
End Sub
```

```

Private Sub cmdSend_Click()
    ' Send data to server
    Call tcpClient.SendData("Client >>> " & txtSend.Text)
    txtOutput.Text = txtOutput.Text & _
        "Client >>> " & txtSend.Text & vbCrLf & vbCrLf
    txtOutput.SelStart = Len(txtOutput.Text)
    txtSend.Text = ""
End Sub

Private Sub tcpClient_Close()
    cmdSend.Enabled = False
    Call tcpClient.Close ' server closed, client should too
    txtOutput.Text = txtOutput.Text & "Server closed connection." & vbCrLf
    txtOutput.SelStart = Len(txtOutput.Text)
End Sub

Private Sub tcpClient_Connect()

    ' When connection occurs, display a message
    cmdSend.Enabled = True
    txtOutput.Text = "Connected to IP address: " & _
        tcpClient.RemoteHostIP & vbCrLf & vbCrLf
End Sub

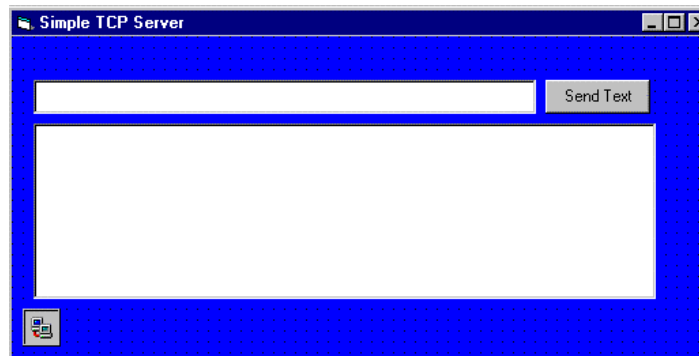
Private Sub tcpClient_DataArrival(ByVal bytesTotal As Long)
    Dim message As String
    Call tcpClient.GetData(message) ' get data from server
    txtOutput.Text = txtOutput.Text & message & vbCrLf & vbCrLf
    txtOutput.SelStart = Len(txtOutput.Text)
End Sub

Private Sub tcpClient_Error(ByVal Number As Integer, Description As String, ByVal Scode As Long,
ByVal Source As String, ByVal HelpFile As String, ByVal HelpContext As Long, CancelDisplay As
Boolean)
    ' If the client fails to connect to server, then this code executes
    Dim result As Integer
    result = MsgBox(Source & ": " & Description & vbCrLf & "Doh! Can't connect to server!", _
        vbOKOnly, "TCP/IP Error")
    ' Source variable is the control (winsock in this case) causing the error
    ' Description variable cites the error message
    ' vbOKOnly is the control button
    ' TCP/IP Error is the MsgBox caption
End
End Sub

```

Step 4: Create the Client – Program and Form name: tcpServerDemo and frmServerDemp respectively. Also make sure Project-Component: **Microsoft Winsock Control 6.0**

Step 5: Client GUI



Step 6: Client code programming

```

' FILE: tcpServer
' DATE: 01/15/00 12:30
' AUTH: P.Oh
' DESC: A Simple TCP Server
' REFS: Deitel p. 826 Fig. 19.10
Option Explicit

Private Sub Form_Load()
    cmdSend.Enabled = False

    ' Set up local port and wait for connection
    tcpServer.LocalPort = 5000
    Call tcpServer.Listen
End Sub

Private Sub Form_Resize()
    On Error Resume Next
    Call cmdSend.Move(ScaleWidth - cmdSend.Width, 0)
    Call txtSend.Move(0, 0, ScaleWidth - cmdSend.Width)
    Call txtOutput.Move(0, txtSend.Height, ScaleWidth, _
        ScaleHeight - txtSend.Height)
End Sub

Private Sub Form_Terminate()
    Call tcpServer.Close
End Sub

Private Sub tcpServer_Close()
    cmdSend.Enabled = False
    Call tcpServer.Close ' client closed, server should too
    txtOutput.Text = txtOutput.Text & "Client closed connection." & vbCrLf & vbCrLf
    txtOutput.SelStart = Len(txtOutput.Text)
    Call tcpServer.Listen ' listen for next connection
End Sub

Private Sub cmdSend_Click()
    ' Send following text data to the client
    Call tcpServer.SendData("Server >>> " & txtSend.Text)
    ' Repeat text data in server's txtOutput.Text window
    txtOutput.Text = txtOutput.Text & "Server >>>" & txtSend.Text & _
        vbCrLf & vbCrLf
    ' Clear the txtSend.Text window
    txtSend.Text = ""
    txtOutput.SelStart = Len(txtOutput.Text)
End Sub

Private Sub tcpServer_ConnectionRequest(ByVal requestID As Long)

    ' Ensure that tcpServer is closed
    ' before accepting a new connection
    If tcpServer.State <> sckClosed Then
        Call tcpServer.Close
    End If

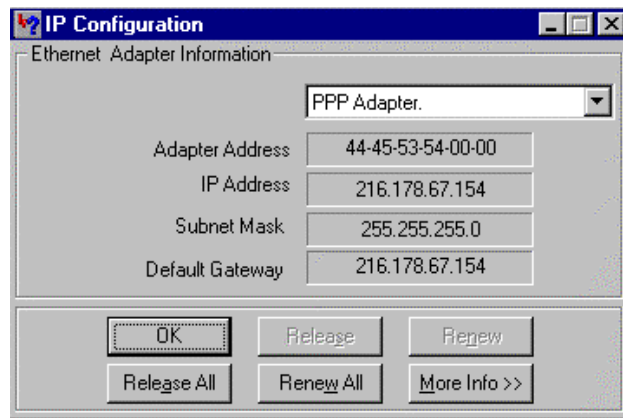
    cmdSend.Enabled = True
    Call tcpServer.Accept(requestID) ' accept connection
    ' Display following message on server application:
    txtOutput.Text = "The connection from IP Address: " & _
        tcpServer.RemoteHostIP & " is successful" & vbCrLf & _
        "Port #: " & tcpServer.RemotePort & vbCrLf & vbCrLf
End Sub

```

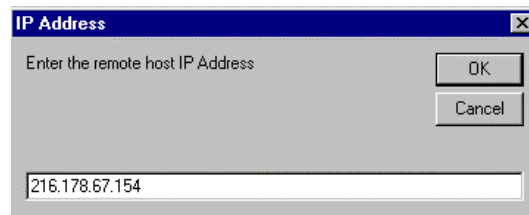
```
Private Sub tcpServer_DataArrival(ByVal bytesTotal As Long)
    Dim message As String
    Call tcpServer.GetData(message) ' get data from client
    txtOutput.Text = txtOutput.Text & message & vbCrLf & vbCrLf
    txtOutput.SelStart = Len(txtOutput.Text)
End Sub

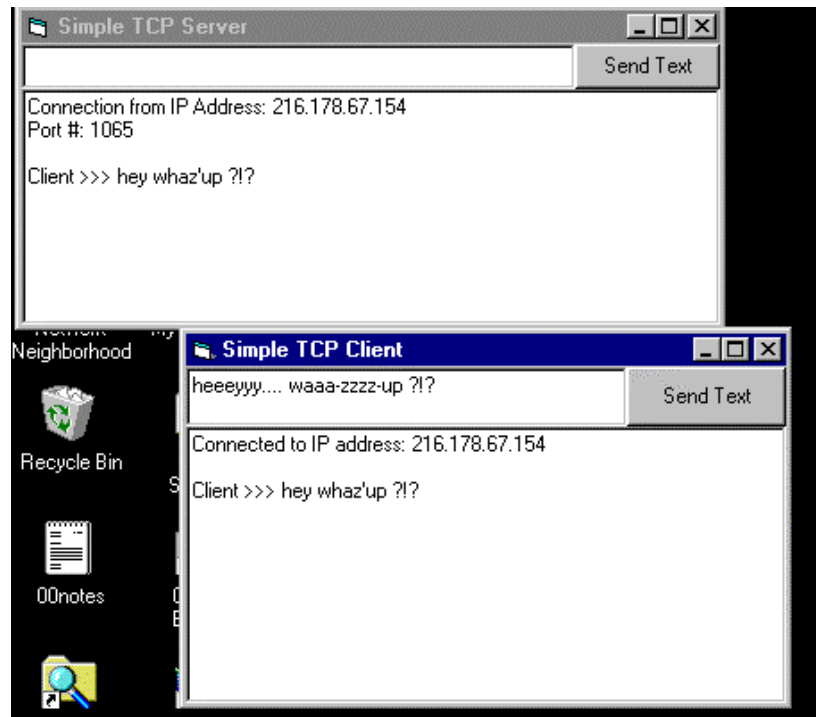
Private Sub tcpServer_Error(ByVal Number As Integer, Description As String, ByVal Scode As Long, ByVal
Source As String, ByVal HelpFile As String, ByVal HelpContext As Long, CancelDisplay As Boolean)
    Dim result As Integer
    result = MsgBox(Source & ": " & Description, _
        vbOKOnly, "TCP/IP Error")
    End
End Sub
```

Step 7: Run Server and note IP address (Start-Run winipcfg)



Step 8: Run Client and type in Server's IP address. Client and Server should be able to communicate





Messages between client and server can be sent and received
A simple chat program!

How does it work?

Server

1. Server must be setup to listen for connections from clients on a specific port number as specified by Winsock control's `LocalPort` property. Values are integers less than 65535 and numbers below 1024 are reserved for system services. Registering the port number is typically done in the `Form_Load` procedure:

```
tcpServer.LocalPort = 5000
```

2. Server is told to listen indefinitely for client's attempt to connect; typically in `Form_Load`

```
Call tcpServer.Listen
```

3. When connection request is received event procedure `ConnectionRequest` executes and an attempt to accept this request is through Winsock's `Accept` method

```
Call tcpServer.Accept(requestID)
```

The value `requestID` was passed as an argument in `ConnectionRequest`. The client-server connection has been established.

4. Messages arriving at server are processed in `DataArrival` procedure. Strings are retrieved from and sent to client using:

```
tcpServer.GetData(message)
tcpServer.SendData(message)
```

5. When transmission is complete and server closes connection, the `Close` procedure is executed

```
Call tcpServer.Close
```

Client

1. `Form_Load` prompts user to type in the server's IP address. `localhost` is a reserved; it's the IP address of the user's computer

```
tcpClient.RemotePort = 5000
Call tcpClient.Connect
```

means that client should attempt a connection to server on port 5000

2. If connection is successful then `tcpClient_Connect` is executed and a short message is displayed
3. If data arrives at client from server, the event procedure `tcpClient_DataArrival` is executed. String is stored in `message`

```
Call tcpClient.GetData(message)
```

4. Clicking `cmdSend` button calls `cmdSend_Click` and text in the `txtSend` box is submitted to server

```
Call tcpClient.SendData("CLIENT >>> " & txtSend.Text)
```

Conclusions:

- Winsock control provides relatively straightforward "Internet" programming.
- Winsock is the foundation of Internet capability in Microsoft's OS, allowing browsers, FTP, chat and terminal applications
- Strings can be sent between client and server through a chat program. Hence we can actuate peripherals or read sensors!
- 8255 can be installed in server. The 8255.dll allows read/write capability and hence clients can read/write to 8255 via the Internet
- Can upload data to a web server and view "real-time" data acquisition with browsers!