Lab 11: Creating Internet-Aware Applications

For background information on this lab, click each one of these topics:

Objectives

By the end of this lab, you will be able to:

- Build a Web browser client by using the WebBrowser control.
- Build a component that provides downloading capabilities.
- Create the client and server code for a peer-to-peer chat application.

Prerequisites

Before starting this lab, you should be familiar with the following concepts:

- Socket technology and the Winsock control
- The File Transfer Protocol (FTP)
- the Hypertext Transfer Protocol (HTP)
- The contents of this chapter

Lab setup

To complete this lab, you will need the following:

- Visual Basic 5.0 or later

The exercises in this lab use Personal Web Server, however, they will also work with any Web server.

To see a demonstration of the completed lab solution, click this icon.

Estimated time to complete this lab: **45 minutes**

Note There are project and solution files associated with each lab. If you installed the labs during Setup, these files are in the folder <*Install Folder*>\Labs on your hard disk. If you did not install the labs during Setup, you can find them in the \Labs folder of the *Mastering Microsoft Visual Basic 5* CD-ROM.

Exercises

The following exercises provide practice working with the concepts and techniques covered in Chapter 11.

Exercise 1: Building the Browser Application

In this exercise, you will use the WebBrowser control to create a browser that is capable of viewing files, HTML pages, and ActiveX documents.

Exercise 2: Building the FTP Download Component

In this exercise, you will build a reusable component that provides FTP downloading services for a client.

Exercise 3: Adding Browser Capabilities to the Component

In this exercise, you will extend the FTPService component in the previous exercise to provide users with the ability to view files on a remote computer and select one for downloading.

Exercise 4: Coding a Chat Session Application

In this exercise, you will create an application that uses the Winsock control to conduct a peer-to-peer chat session with another application on the same or remote computer.

Exercise 1: Building the Browser Application

In this exercise, you will use the **WebBrowser** control to create a browser that is capable of viewing files, HTML pages and ActiveX documents.

You begin by starting a new project, and then creating a toolbar for the browser. You will also create a status bar and add Web browser navigation capabilities. Finally, you will implement a progress bar within the status bar.

Creating the User Interface

In this first procedure, you will create the initial project and set up the user interface components, as shown in the following illustration.

▶ **Create the initial project**

1. Create a new Standard EXE.
2. In the main form, add **Toolbar**, **Statusbar**, and **ProgressBar** controls. These controls are all located in the Microsoft Windows Common Controls 5.0 component.
3. For the **ProgressBar** control, set the **Visible** property to **False**.
4. Add a **WebBrowser** control to the main form. This control is located in the Microsoft Internet Controls component.
5. Save the project in the folder *<install folder>*\Lab11.

▶ **Design the toolbar**

1. Add a drop-down combo box to the toolbar that enables users to type in new URLs and store past URLs. Add an appropriate label for the combo box.
2. To the right of the combo box, add five command buttons to the toolbar. Create the buttons as a control array, providing each one with the same name and incrementing the **Index** property.

Command buttons added through the property page for the toolbar are always added starting at the left edge of the toolbar. This will interfere with the combo box you have just added. For this reason, add the command buttons from the command toolbox.

Initialize the controls based on the information in the following table.

Name	Picture	Purpose
cmdStop	Trffc14.ico	Stop an asynchronous Web Browser operation.
cmdGoBack	Trffc04.ico	Go to the prior page as displayed in the Web Browser
cmdGoForward	Trffc02.ico	Go to the next page as displayed in the Web Browser
cmdHome	House.ico	Go to the home page, as specified in the registry.
cmdSearch	Binoculr.ico	Go to the Web search page, as specified in the Registry.

▶ **Design the status bar**

1. Set the **AutoSize** property of the first panel to **sbrContent**. This will create the size of the panel based on the size of the text it contains, but with the minimum width specified by the **MinimumWidth** property. This panel will be the location of the **ProgressBar** control when it is displayed.
2. Insert a second panel to display the title of the current page displayed in the Web Browser. Set its **AutoSize** property to **sbrSpring** so that its size is based on the width of the status bar.

Coding the Application

In the next procedure, you will add the underlying code for the application. You will also learn how to use a status bar panel to display a progress bar, and how to bring other application windows into the foreground.

Note The following instructions do not specify the addition of error handlers. However, many of the operations performed by this application may fail, such has navigating to an invalid URL. It is strongly suggested that you provide error handling throughout this application.

▶ **Initialize the WebBrowser control**

1. Implement the Start Page code.
   a. Declare a form-level string variable to hold the name of the Start Page.
   b. In the Form_Load event, use the **GetSetting** statement to extract the default Start Page from the registry. Use the parameter values listed in the following table.

Parameter	Value
AppName	IAwareApp
Section	Startup
Key	StartPage
Default	HomePage.HTML in the application's directory (see App.Path).

   c. Use the **Navigate** method of the **WebBrowser** control to display the Start Page.
   d. In the Form_Unload event, use the **SaveSetting** statement to save the name of the Start Page to the registry.
2. Size the **WebBrowser** control based on the size of the form.
   a. Add a handler for the Form_Resize event.
   b. Size the **WebBrowser** control so that it fills the display area of the form, between the **Toolbar** control and the **StatusBar** control.

It is not necessary to size the form in the Form_Load event because the Form_Resize event occurs just before the form is initially displayed.

▶ Add WebBrowser navigation

1. Implement the combo box for the URL.
   a. Add a handler for the combo box KeyPress event.
   b. If the KeyPress value is a carriage return (ASCII value 13) and the text in the Edit field of the combo box is not a null string, use the **Navigate** method of the **WebBrowser** control to display the URL specified, and use the **AddItem** method to add the URL to the combo box list.
   c. Add a handler for the combo box Click event.
   d. In the handler, navigate the **WebBrowser** control to the URL selected in the combo box.
2. Implement the navigation command buttons.
   a. In the Click event of the **Stop** button on the toolbar, stop the current operation and return to the prior URL if the Web Browser is busy (refer to the **Busy** property).
   b. In the Click event of the **Back** button, invoke the **GoBack** method of the Web Browser. Repeat this step for the **Forward**, **Home**, and **Search** buttons by using their associated methods.

▶ Implement the ProgressBar control

1. Make the **Statusbar** control the parent of the **ProgressBar** control. This will enable the **ProgressBar** control to be displayed on top of the **StatusBar** control, and to be located based on the **StatusBar** control coordinates.
   a. Add a standard module to the project.
   b. In the standard module, add a declaration statement for the **SetParent** API. Because this is a Win32 function, you can use the API Text Viewer that ships with Visual Basic to add the correct **Declare** statement.
   c. In the Form_Load event, set the parent of the **ProgressBar** control window to the **StatusBar** control window by using the **SetParent** API.
2. Display and update the **ProgressBar** control during a download operation.
   a. Add a handler for the DownloadBegin event of the **WebBrowser** control.
   b. Initialize the **ProgressBar** control to have a minimum value of 0, a maximum value of 100, and a starting value of 0.
   c. Move the **ProgressBar** control to fill the first panel of the **StatusBar** control, and set its **Visible** property to **True**.
3. Hide the **ProgressBar** control when downloading is complete.
   a. Add a handler for the DownloadComplete event of the **WebBrowser** control.
   b. Set the **Visible** property of the **ProgressBar** control to **False**.
4. Update the **ProgressBar** control during the download operation.
   a. Add a handler for the ProgressChange event of the **WebBrowser** control.
   b. If the **Progress** parameter is not –1, and the **ProgressMax** parameter is not 0, set the **Value** property of the **ProgressBar** control to percentage that the operation is complete.

Percent complete is calculated as **Progress** times 100, divided by **MaxProgress**.

▶ Use InternetExplorer to extend context-sensitive Help

1. Add a menu commands to the form, as listed in the following table.

File	**Options**	**Help**
Open	**Set Start Page**	**MS on the Web >**
(separator)		**Home Page**
Exit		**Developer Only**
		Visual Basic
		About

2. Enable Internet Explorer to display the **MS on the Web** sites.
   a. Add a private form-level object variable of type **InternetExplorer**.
   b. Add a handler for the **MS on the Web** menu commands.
   c. In the handler, check to see if an instance of the **InternetExplorer** control is running. If not, create a new instance, as shown in the following code:

```
If ie is Nothing then
    Set ie = New InternetExplorer
End If
```

   d. Use the **Navigate** method of **InternetExplorer** control to display an appropriate Web site based on the following table.

Site	URL
Home Page	http://www.microsoft.com
Developers Only	http://www.microsoft.com/devonly
Visual Basic	http://www.microsoft.com/VBasic

   e. Set the **Visible** property of the InternetExplorer control to **True**.
3. Make the **InternetExplorer** control the topmost application.
   a. In the standard module, add a declaration for the Win32 API **SetForegroundWindow** function.
   b. After the statement that makes the **InternetExplorer** control visible, add a call to **SetForegroundWindow**.

▶ **Finish coding the application**

1. Implement the **Set Start Page** menu command to enable users to specify a different Start Page.
2. Implement the **Open** menu command that displays a dialog box to let users specify and navigate to a URL.
3. Implement the **Exit** menu command by unloading the form.

Do not use the **End** statement in the event procedure of the **Exit** menu item because the Unload event will not occur and the reference to the startup page will not be saved to the registry.

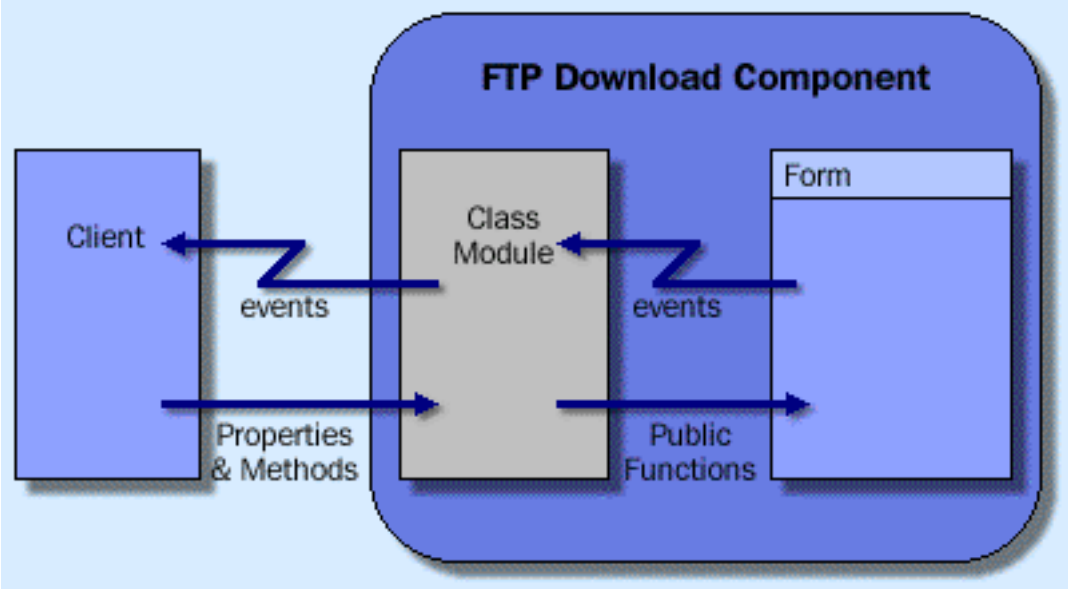
Exercise 2: Building the FTP Download Component

In this exercise, you will build a reusable component that provides FTP downloading services for a client.

You will create a class module that interacts with clients, and a form that provides users with a visual indication of the download progress and the ability to cancel anytime before completion.

You will also use the **Internet Transfer** control to work with FTP, and generate events from forms.

The following illustration shows the architecture of the completed component.



▶ Create project

1. Create a new ActiveX DLL project.
   a. Set the project name to FTPDownload.
   b. Set the class name to clsDownload.
   c. Set the instance of the class to MultiUse.
2. Add the form.
   a. Add a new form to the project.
   b. Set the name of the form to frmFTPCopy.
   c. Set the **Caption** property to FTP Download.

▶ Implement the frmFTPCopy form

1. Add the user interface.
   a. Add a label that displays the source and destination.
   b. Add a **Cancel** command button.
   c. Add an **Internet Transfer** control to the form (the location is not important).
2. Declare events for the form.
   a. Declare a CopyCancelled event.
   b. Declare a CopyCompleted event.
   c. Declare a CopyError event with two parameters, an integer for an error code and a string for an error description.
3. Implement a public entry point.
   a. Create a public function named **BeginCopy**. This function should take three parameters: the URL of the FTP service, and the file names of the source and the destination.
   b. Make the form visible.
   c. Use the **Execute** method of the **Internet Transfer** control to begin an FTP **Get** operation, specifying the source and destination.

   The following code shows the form of the statement:

```
sOperation = "" & " Get " & Source & " " & Destination & ""
INet1.Execute URL, sOperation
```

d. Following the **Execute** call, enter a loop that calls **DoEvents** until the **StillExecuting** property of the **Internet Transfer** control returns a value of **False**.

4. Implement the StateChanged event handler.
   a. Add the StateChanged event handler
   b. If the state the state of the copy is **icResponseCompleted**, unload the form and raise the CopyCompleted event.
   c. If the state is **icError**, unload the form and raise the CopyError event. Pass the **ResponseCode** and **ResponseInfo** property values of the **Internet Transfer** control as the error code and error description parameters, respectively.
5. Implement the **Cancel** button.
   a. Add a handler for the Click event of the **Cancel** command button.
   b. Call the **Cancel** method for the **Internet Transfer** control.
   c. Unload the form.
   d. Raise the CopyCancelled event.

▶ **Implement the clsDownload class module**

1. Create an instance of the frmFTPCopy form.
   a. Declare an object variable of type frmFTPCopy that allows the **clsDownload** class to receive events. (Refer to the **WithEvents** declarator.)
   b. In the Initialize event handler for the class, create a new instance of the frmFTPCopy form.
   c. Declare the three public events CopyCompleted, CopyCancelled, and CopyError to match those of the frmFTPCopy form, using copy and paste.
   d. Implement a handler for each of the form events, and raise the associated class event back to the client, as shown in the following code:

```
Private Sub frmFTPCopy_CopyCompleted()
    RaiseEvent CopyCompleted
End Sub
```

2. Add the client interface.
   a. Add a public function named Copy, which takes three string parameters (URL, Source, and Destination) and returns a **Boolean** value that indicates success or failure.
   b. Validate that the URL begins with the prefix FTP://. If not, fix the URL appropriately.
   c. Call the BeginCopy function in the form, passing the appropriate parameters.

Build and Test the Component

In the next procedure, you will build the FTPDownload component, and update the client so that it can use the FTP service.

There are also two optional activities that you can do: You can add a function that validates the destination folder, and you can provide users with a progress indicator during the download.

▶ **Build the FTPDownload.dll component**

1. Create the DLL component by clicking the **Make DLL** command on the **File** menu.
2. Run the component so that it can be debugged. There is no visible representation of the component until it is called by the client.

▶ **Update the client to use the FTPDownload service**

1. Provide the download user interface.
   a. Load the client application created in the first exercise into a separate instance of Visual Basic.

b. Add a reference to the FTPDownload component. Be sure that the reference is to the running instance of the component (the .vbp file), rather than to the DLL.

c. Add a new form to the project named frmDownload.

d. In the form, add labels and text boxes for users to provide the URL, and the file names for the source and destination.

e. Add the **Copy** and **Cancel** command buttons.

2. Implement the client's FTPDownload capability.

a. In the code of the new form, declare a form-level object variable of type **FTPDownload** that can receive the component's events.

b. In the form's Load event, create an instance of the FTPDownload component.

c. Implement a handler for the **Copy** command button's Click event that invokes the **Copy** method of the **FTPDownload** object.

d. Implement a handler for the **Cancel** command button's Click event that causes the form to be unloaded.

e. Add a subroutine, **EnableCopyButton**, that enables the **Copy** command button only if all three text boxes contain user-entered text.

f. Add a handler for the Change event of the URL text box that calls the **EnableCopyButton** subroutine. Repeat this step for the source and destination text boxes.

g. Add an event procedure for the component's CopyCompleted event that displays a message box informing users about the outcome of the FTP download operation. Repeat this step for the CopyCancelled and CopyError events.

h. On the **File** menu of the main form, add a **Transfer...** command.

i. Add a handler for the **Transfer** menu item that shows the FTPDownload form.

3. Build and test the FTPDownload capability.

Extending the Component (Optional)

This is the first of two optional activities that have you add functionality to your component. First, add a function that validates the destination folder. Second, add a progress indicator to the service.

▶ Add a function that validates the destination directory

Two attributes of the destination should be addressed. In this section, you will ensure that the destination directory exists and prevent copying over a destination file should it already exist.

1. Implement the destination validation function in the clsDownload class module.

a. Add the private function **ValidDestinationPath** that takes a destination string as a parameter and returns **True**, if the destination is a valid path, or **False** if it is not valid.

b. Extract the path of the destination string.

c. Using the **Dir$** function, test whether or not the path is valid, and return the appropriate value.

2. Implement the **ReplaceFile** function. In this function, test for the existence of the destination file. If it already exists, display a message to users that asks whether or not they want to replace the file. If they answer Yes, rename the exiting file with a .bak extension.

3. In the **Copy** function, invoke the **ReplaceFile** and **ValidDestinationPath** functions before calling the **BeginCopy** function of the frmFTPCopy form.

4. Update the CopyCancelled and CopyError events to restore the .bak file to its original name, if one was created.

▶ Provide user feedback during download

One problem with the download service is that users get no feedback that the operation is still underway. In this section, you will implement the flying icon similar to that displayed when copying files with Windows Explorer.

1. Install the GlobalTimer component located in the folder\Labs\Lab11\FTP\GTimer.

2. Update the user interface.
   a. Scale the frmFTPCopy form of the component to a width and height of 100 units.
   b. Place the icon Files02b.ico in an image control at position (10, 50).
   c. Place the icon Openfold.ico in an image control at position (80, 20).
   d. Place the icon Drag1pg.ico in an image control anywhere on the form.
3. Implement the flying icon.
   a. Although the **Internet Transfer** control provides notification on the download process through the State_Changed event, it is not consistent enough to display the flying icon. For this reason, use the Timer component included with the *Mastering Microsoft Visual Basic 5.0* CD-ROM, and add a reference to the GlobalTimer component.
   b. Declare a form-level object variable of type **CTimer** that allows receipt of object events.
   c. Declare a form-level variable, FlyingImageX, of type **Single** that holds the current X position of the flying icon.
   d. Declare the form-level constants listed in the following table.

Name	Type	Value
interval	**Single**	5
FlyingImageStart	**Single**	20
FlyingImageEnd	**Single**	80

   e. In the Form_ Load event, create an instance of the **Timer** object.
   f. Add a private function, **FlyingImageY**, that does not take any parameters and returns the type **Single**.
   g. In the handler, calculate the current Y position of the flying icon and increment the current value of the X position, as shown in the following code:

```
Private Function FlyingImageY() As Single
    FlyingImageY = 1 / 36 * (FlyingImageX - 50) * _
                   (FlyingImageX - 50) + 20
    FlyingImageX = FlyingImageX + interval
    If FlyingImageX > FlyingImageEnd Then
        FlyingImageX = FlyingImageStart
    End If
End Sub
```

   h. Add a private **Sub** routine named FlyImage that calls the **FlyingImageY** function, and moves the flying image control based on the current values of the X and Y positions.
   i. Add a handler for the TimerEvent event to the **Timer** control that calls FlyImage.
   j. In the StateChanged event, if the state is icConnected, initialize the FlyingImageX variable to the FlyingImageStart constant, and call the flying image (FlyImage) once. Follow this with a call to start the timer with an interval of 200.
   k. If the event is icResponseCompleted or icError, stop the timer.
   l. In the Click event of the **Cancel** command button, stop the timer.
4. Recompile and test the component.

Exercise 3: Adding Browser Capabilities to the Component

In this exercise, you will extend the FTPDownload component to provide users with the ability to view the files on a remote computer, and selecting a file for downloading.

You will create the component so that it lists only files in the main FTP folder of the server. Optionally, you can modify the component to provide the ability to navigate through the folder structure.

▶ Create the user interface

1. Add a second form to the component.
2. In the form, add a list box to display the list of available files, and the command buttons **OK** and **Cancel**.
3. Place a copy of the Internet Transfer Control anywhere on the form.

▶ Implement the frmFTPGetFileName form

1. Declare three events: FileGetCompleted, FileGetCancelled, and FileGetError.
2. Add the public entry subroutine, BeginGetFile, which takes a URL string as a parameter. In this routine, execute an FTP Directory operation by using the URL parameter.
3. Add a handler for the StateChanged event. If the event is icResponseCompleted, use the file names in buffered data (see **GetChunk**) to fill the list box. Names are separated by carriage return/linefeed pairs and folders are signified by a forward slash (/) in the right-most position of the name. If the event is icError, raise the error back to the form's client, and unload the form.
4. Add a handler for the **OK** command button. The handler should raise the appropriate event, and pass the selected file name back to the client, and unload the form.
5. Add a handler for the **Cancel** command button that cancels any executing FTP operations, raises the appropriate event, and unloads the form.
6. Add a handler to resize the list box as appropriate based on the size of the form.

▶ Update the class module

1. Declare a private object variable to hold a pointer to an instance of the form frmFTPGetFileName. Be sure that it allows you to receive events.
2. Declare three events to pass to the component's clients, one for each of the events to be received from the frmFTPGetFileName form.
3. Add a public subroutine, **GetFileName**, which receives the URL as a parameter from the client.
4. Create an instance of the frmFTPGetFileName form, if one does not exist. Ensure that the URL has the prefix FTP://, and invoke the BeginGetFile routine in the form.
4. Add a handler for each of the events raised by the frmFTPGetFileName form, and pass the associated class event to the client.

▶ Modify the client to use the revised component

The final step is to modify the client application to take advantage of the new component functionality.

1. Update the FTP Download form to include a command button labeled **Get File**.
2. Add code to enable the **Get File** command button, only if the URL text box contains user-entered text.
3. In the handler for the Click event of the **New** button, invoke the **GetFileName** method of the **FTPDownload** object.
4. Add handlers for the new component events. If the event is GetFileCompleted (or the equivalent), fill the **Source** text box with the selected file name.
5. Build and test the component.

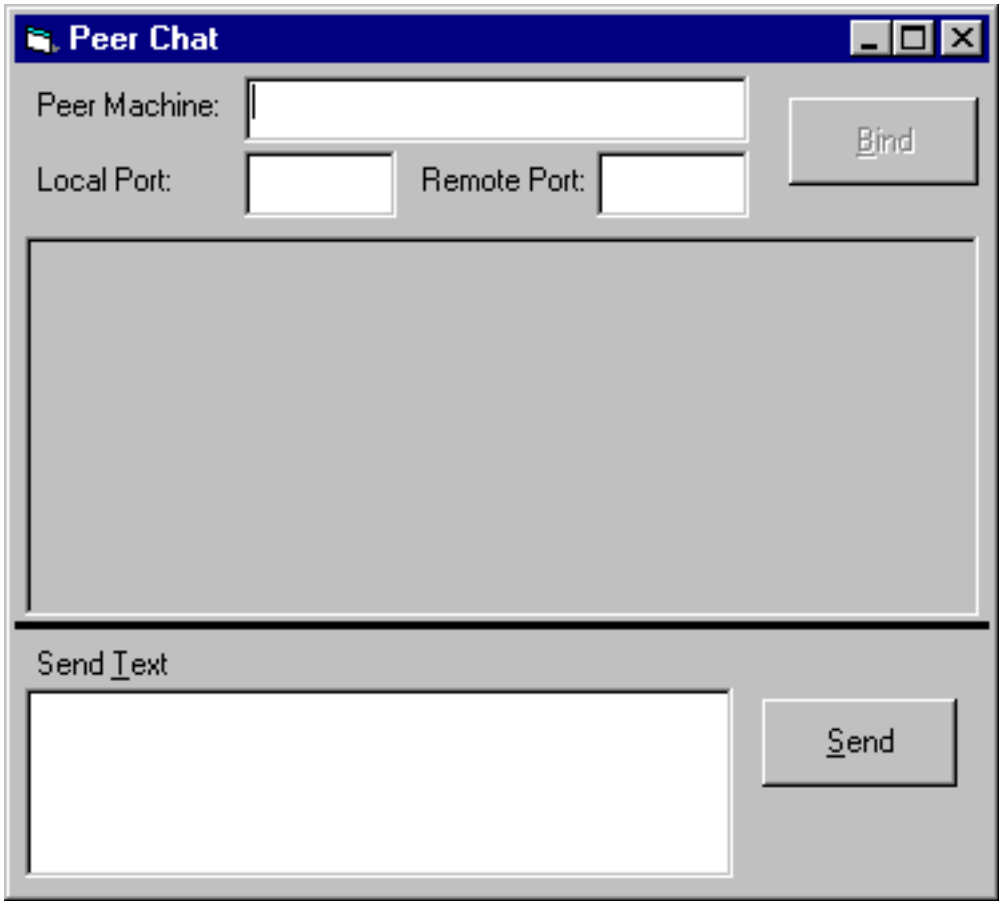
Exercise 4: Coding a Chat Session Application

In this exercise, you will create an application that uses the **Winsock** control to conduct a peer-to-peer chat session with another application on the same or remote computer.

▶ Create the Project

1. Create a new Standard EXE project.
2. Add the Winsock component to the project.

3. Add the **Winsock** control to the form.
4. Modify the form to resemble the following illustration.



5. Save the project as MyChat in the folder \Lab11.

▶ Implement binding

In the event handler for the binding Click event:

1. Set the **RemoteHost** property of the **Winsock** control.
2. Set the **RemotePort** property of the **Winsock** control.
3. Use the **Bind** method to set the local port.

▶ Implement data sending

1. Add a handler for the **Send** button's Click event.
2. Use the **SendData** method of the **Winsock** control to send data from the appropriate text box.
3. Clear the text in the text box.

▶ Process received data

1. Add a handler for the **Winsock** control's Data_Arrival event.
2. If data exists, receive it from the control's buffer and add it to the beginning of the **Received Data** text control. Store only the last 2KB of the existing received data.

▶ Enable the bind button

1. Set the **Enabled** property of the **Bind** button to **False**.
2. Add a private procedure that enables the **Bind** button only if the controls on the remote computer, remote port, and local port contain data.
3. To call the added procedure, add an event handler for the Change event of the remote computer, remote port, and local port.

▶ **Test the application**

1. Compile the application.
2. Run the application in the Visual Basic environment.
3. Set the Remote Computer name to the network name of your computer.
4. Set the Remote Port to 1000.
5. Set the Local Port to 1001.
6. Bind the application.
7. Run a copy of the compiled application on the same computer.
8. Set the Remote Computer name to the network name of your computer.
9. Set the Remote Port to 1001.
10. Set the Local Port to 1000.
11. Bind the application.
12. Send text from each port, and validate that it is properly received.