

Chapter 7

Components

Learning Objectives

After completing this chapter, you will be able to:

- *Work with BackgroundWorker component.*
- *Work with ErrorProvider component.*
- *Work with EventLog component.*
- *Work with FileSystemWatcher component.*
- *Work with HelpProvider component.*
- *Work with ImageList component.*
- *Work with MessageQueue component.*
- *Work with PerformanceCounter component.*
- *Work with ServiceController.*
- *Work with Process component.*
- *Work with Timer component.*



INTRODUCTION

In this chapter, you will learn about various components used in VB.NET and their implementation. These components are discussed next.

BackgroundWorker Component

The **BackgroundWorker** component is used to execute the time-consuming operations of the applications asynchronously in the background. To add the **BackgroundWorker** component to a form; double-click on it in the toolbar. Alternatively, drag and drop it on the form, it will be added to the component tray. You can view its properties in the **Properties** window. The **BackgroundWorker** component has four events, **Disposed**, **DoWork**, **ProgressChanged**, and **RunWorkerCompleted**. These events are discussed later in this chapter with the help of an application.

ErrorProvider Component

The **ErrorProvider** component is used to validate the user's input on a control. It is an indication of an error associated with a control on a form. The **ErrorProvider** component displays an error icon next to the target control. The default error icon is indicated by an exclamatory mark, which is red in color. When a user moves the cursor over the icon, a tooltip with the error message strip gets displayed. To add the **ErrorProvider** component to the form; double-click on it in the toolbar. Alternatively, drag and drop it on the form; it will be added to the component tray. You can view its properties in the **Properties** window. An example of the implementation of the **ErrorProvider** component is given below:

Step 1

Start a new project. Add an **ErrorProvider** component, a **TextBox** control, and a **Button** control to the form.

Step 2

Double-click on the **Button** control and enter the following source code to the code window:

```
If Not IsDate(TextBox1.Text) Then
    ErrorProvider1.SetError(TextBox1, "Not a valid date.")
Else
    ErrorProvider1.SetError(TextBox1, "")
End If
```

In the above given source code, if you enter an invalid date in the **TextBox** control and choose the **Button1** control; an error icon will be displayed next to the **TextBox** control, as shown in Figure 7-1. If you move the cursor over the icon; the error message **Not a valid date.** will be displayed.

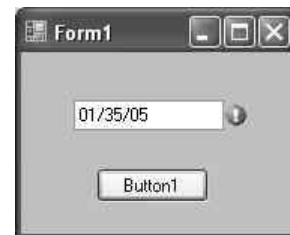


Figure 7-1 The form with the error icon

EventLog Component

Event logging is a standard method with which an application in your system keeps a record of some software and hardware related events. With the help of this record, the system administrator can determine the cause of any hardware or software error in the system. Also, it tries to recover any lost data and prevent the errors from occurring again and again. The **EventLog** component is used to determine the conditions due to which an error has occurred. With the help of this component, you can connect the event logs both to local and remote computers. You can write the entries for these logs and also read the entries from the existing logs. To add the **EventLog** component to your form, double-click on it in the toolbar. Alternatively, you can drag and drop it on the form; it will be added to the component tray. You can view its properties in the **Properties** window.

You can also view the list of event logs of the servers that you have accessed. You can also add the event logs from the list to your form. To view the list, choose **Server Explorer > Servers > Event Logs**. Next, drag an event log from the list and drop it on the form; an instance of the added event log will be created. There are three event logs that are available on every system, **System**, **Security**, and **Application**. The **System** event log tracks the events that occur on the system component such as drivers and services. The **Security** event log tracks the security changes, user log-in, and so on. The **Application** event log tracks the events that occur in the registered application.

Writing an Event Source

The concept of event source is used in the event log system. The event source is used to determine whether an application can write an event log entry and the events with which those entries should associate. Before writing an event log, your application must register itself to the existing event source or create a new event source, which is given below:

Creating an Event Log

You can use the **EventLog.CreateEventSource()** method to create a new event source that will be associated with an event log. The following source code is used to check if an event source “App” exists in the application. If the event source does not exist in the application, then this code will create a new event source.

```
If Not EventLog.Exists("App") Then  
    EventLog.CreateEventSource("App", "Application")
```

FileSystemWatcher Component

The **FileSystemWatcher** component is connected to the directories and it monitors the changes in the directories or the files contained in it. The changes in the directories include creation of new files, addition of subdirectories, and renaming of subdirectories or files. The **FileSystemWatcher** component basically conveys the information about the changes made in the file system by raising relevant events. The **FileSystemWatcher** component can view the files on a local system, a network, or a remote computer. Note that it can view the changes within a directory, but not the changes made in the attributes of the root directory. For example, if you are working in a directory called **C:\My Pictures**, the component will view the changes within the **My Pictures** directory and not in the root directory, **C:.**

Some Important Properties of FileSystemWatcher Component

Some important properties of the **FileSystemWatcher** component are as follows:

EnableRaisingEvents

The **EnableRaisingEvents** property determines whether the component is active or not.

Filter

The **Filter** property is a string value, which is used to determine the files that are being viewed or monitored in a directory. Its default value is *.*. If you want to view the changes in all the files, then set the value of the **Filter** property to "" (empty string).

IncludeSubdirectories

The **IncludeSubdirectories** property determines whether the subdirectories within the specified directory should be viewed or not.

NotifyFilter

The **NotifyFilter** property determines the type of changes to be viewed.

Path

The **Path** property determines the path of the directory to be viewed.

HelpProvider Component

The **HelpProvider** component provides the pop-up help or the online help for various controls, when the F1 key is pressed. With the help of the **HelpProvider** component, an HTML file is displayed with the linked topics. Help is associated with the **HelpProvider** component using the **HelpNamespace** property. You can use the **SetHelpString** method of the **HelpProvider** component to associate a specific help string with other controls, such as **Button** control. The string is associated with a control with the help of the **SetHelpString** method. It is displayed when you press the F1 key while the control is in focus. Similarly, you can use the **SetHelpKeyword** method to determine the help keyword associated with a control and the **SetHelpNavigator** method to determine the help in the form of table of contents, index, and so on. Also, the **HelpProvider** component provides additional properties to a control on the form. These properties are discussed next.

HelpKeyword

It determines whether the **Help** keyword will be associated with the control or not.

HelpNavigator

It determines the constant that indicates the element of the help file to be displayed. It provides seven values to navigate **Topic**, **TopicOfContents**, **Index**, **Find**, **AssociatedIndex**, **Keywordindex**, and **TopicId**.

HelpString

It determines the **Help** string associated with the control.

ImageList Component

The **ImageList** component is used to store the collection of images. These images can be displayed with the help of different controls. To add the **ImageList** component to the form, drag and drop it on the form; it will be added to the component tray. To view some of the important properties of the **ImageList** component, choose the forward arrow on the top right corner of the **ImageList** component, as shown in Figure 7-2.

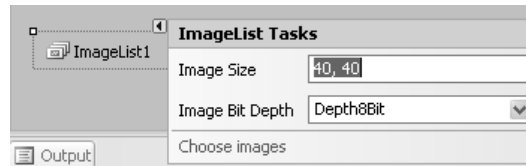


Figure 7-2 Some important properties of the **ImageList** component

Some important properties of the **ImageList** component are discussed next.

Image Size

The **Image Size** property is used to set the size of the image to be displayed.

Image Bit Depth

The **Image Bit Depth** property is used to change the bit depth of images that are imported to the form.

Choose Images

The **Choose Images** property is used to add images to the image collection list of the **ImageList** component. To add an image to the **ImageList** control, choose the **Choose images** property from the Smart Tags or choose **Images** from the **Properties** window; the **Images Collection Editor** dialog box will be displayed, as shown in Figure 7-3. Choose the **Add** button; the **Open** dialog box will be displayed. Select the image that you want to add to the **ImageList** component; the image will be added in the **Images Collection Editor** dialog box. Choose the **OK** button from the **Images Collection Editor** dialog box. You can use the **ImageList** component with all the controls that have the **ImageList** property.

MessageQueue Component

The **MessageQueue** component is used to introduce a message-based communication within an application. With the help of this component, an application can send and receive messages in a queue on the **Message Queuing** servers, create and delete queues, explore the existing queues, and so on. The **MessageQueue** component basically provides a central place to store and remove the data. To add the **MessageQueue** component, double-click on it in the toolbox. Alternatively, you can drag and drop it on the form. Note that this component will be added in the component tray only if you have message queue configured on your system. If an error message is displayed while adding this component to the form, it means that the message queue is not configured on your system. To configure the message queue, follow the steps given next.

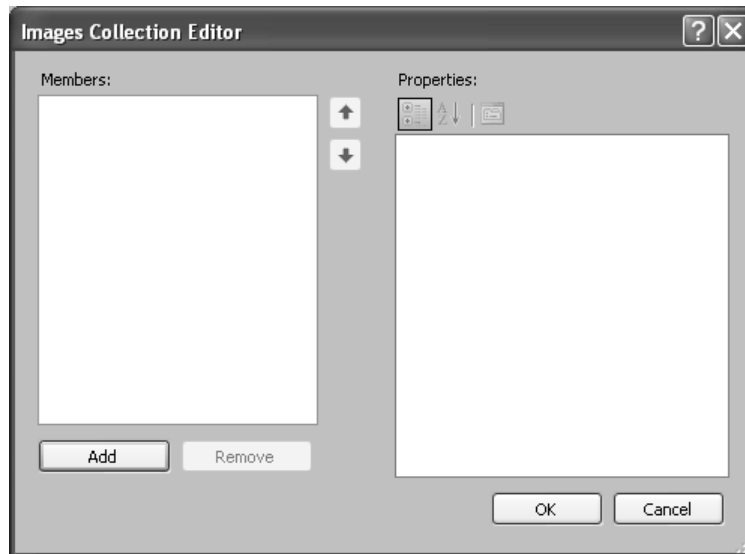


Figure 7-3 The Images Collection Editor dialog box

1. Choose **Start > Settings > Control Panel**; the **Control Panel** window will be displayed, as shown in Figure 7-4.

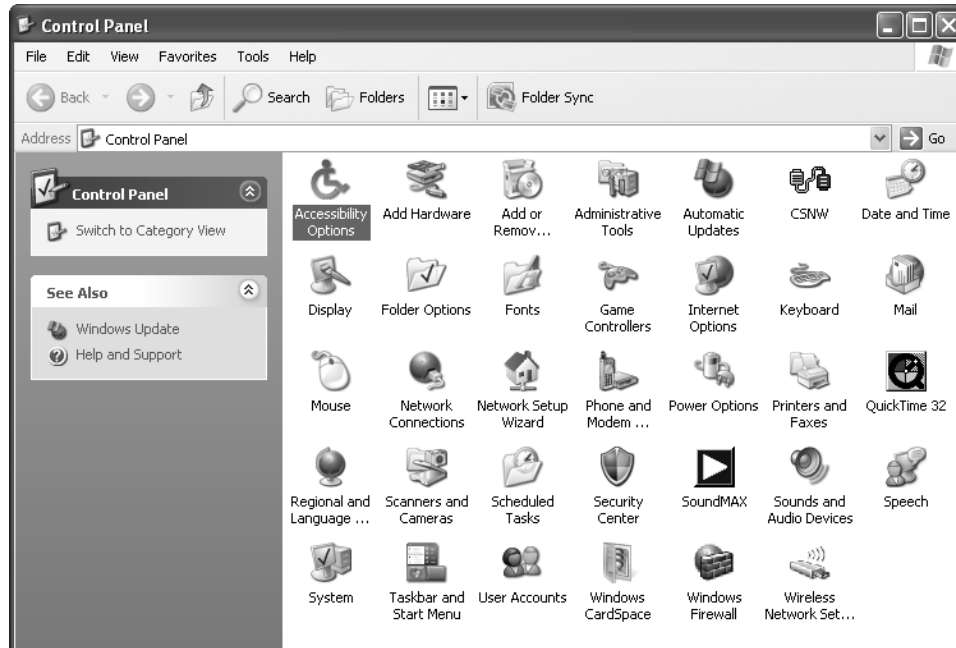


Figure 7-4 The Control Panel window

2. Double-click on the **Add or Remove Programs** icon; the **Add or Remove Programs**

window will be displayed, as shown in Figure 7-5. Choose the **Add/Remove Windows Components** button on the left of the window; the **Windows Components Wizard** dialog box will be displayed, as shown in Figure 7-6.

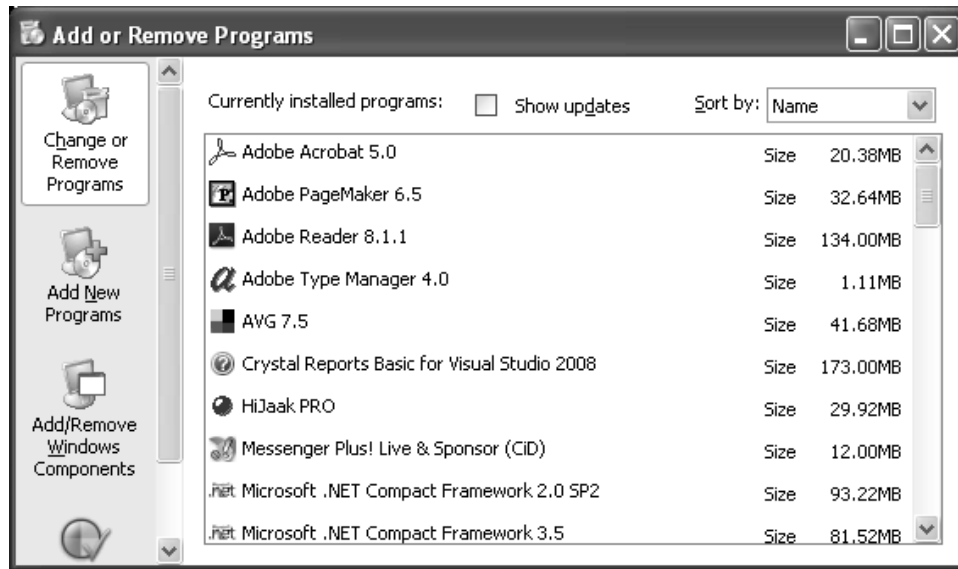


Figure 7-5 The Add or Remove Programs window

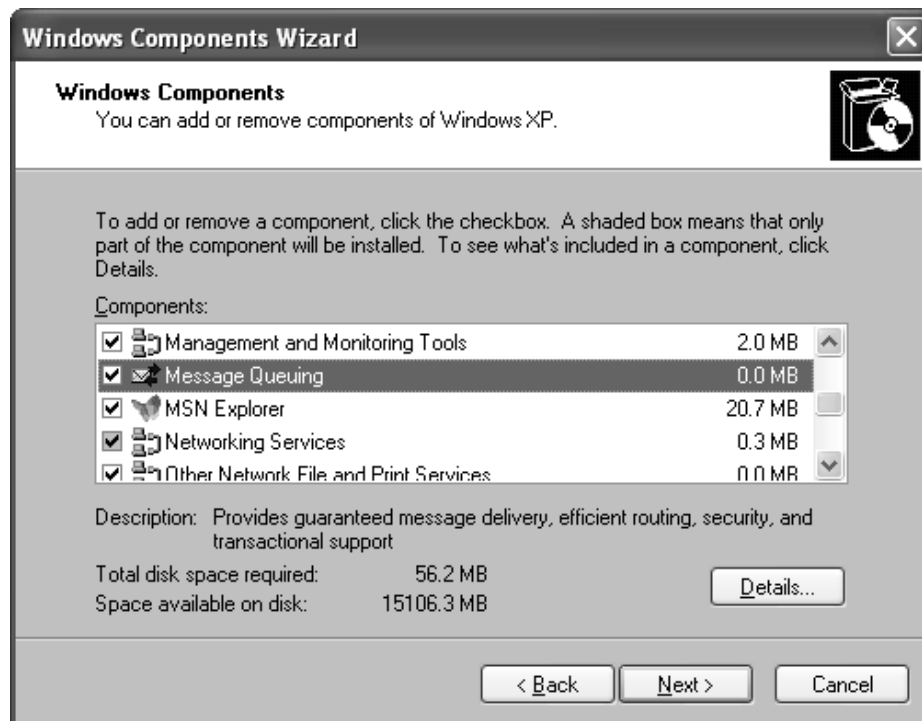


Figure 7-6 The Windows Components Wizard dialog box

3. Select the **Message Queuing** check box from the **Components** area.
4. Choose the **Details** button in the dialog box; the **Message Queuing** dialog box will be displayed, as shown in Figure 7-7. Select all the check boxes in the **Subcomponents of Message Queuing** area. Choose the **OK** button; it will return to the **Windows Components Wizard** dialog box.

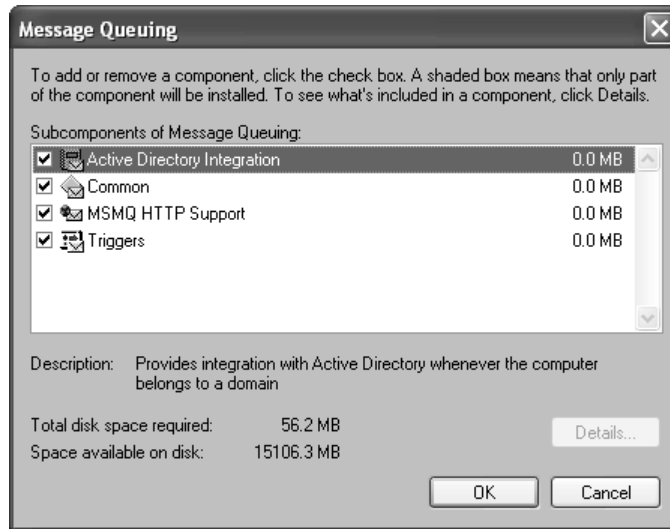


Figure 7-7 The Message Queuing dialog box

5. Choose the **Next** button in the **Windows Components Wizard** dialog box; it will prompt you to insert the CD of the operating system, which is already installed on your system. Insert the CD. After the configuration is completed, choose the **Finish** button.

There are four types of queue in the **MessageQueue** component, which are as follows:

Outgoing

The **Outgoing** queue is used to store the messages temporarily before they are sent to their destinations.

Public

The **Public** queue can be used by different application on different servers within a network.

Private

These type of queues can be used only by the local server.

System

It contains the messages sent from the system and the dead messages. These dead messages are not meant for delivery.

PerformanceCounter Component

The **PerformanceCounter** component is used to view the behavior of the performance objects such as processors, memory, disks, and so on. There is a specific category for all the performance counters. For example, the **Available Bytes** and **Cache Bytes** counters fall under the **Memory** category. You can create your own category and then specify the counters. There are two ways of creating the category, using the **Server Explorer** and through coding. In this section, you will learn to create it using the **Server Explorer** only.

Creating New Category of Performance Counter Using Server Explorer

To create a new category of performance counters, choose **Server Explorer > Servers > Computer name > Performance Counters** from the design window of the current application. Right-click on **Performance Counter** and then choose **Create New Category**; the **Performance Counter Builder** dialog box will be displayed, as shown in Figure 7-8. You can enter the values for all the parameters according to your requirement and then, choose the **OK** button; the new category will be added to the **Performance Counter** list.

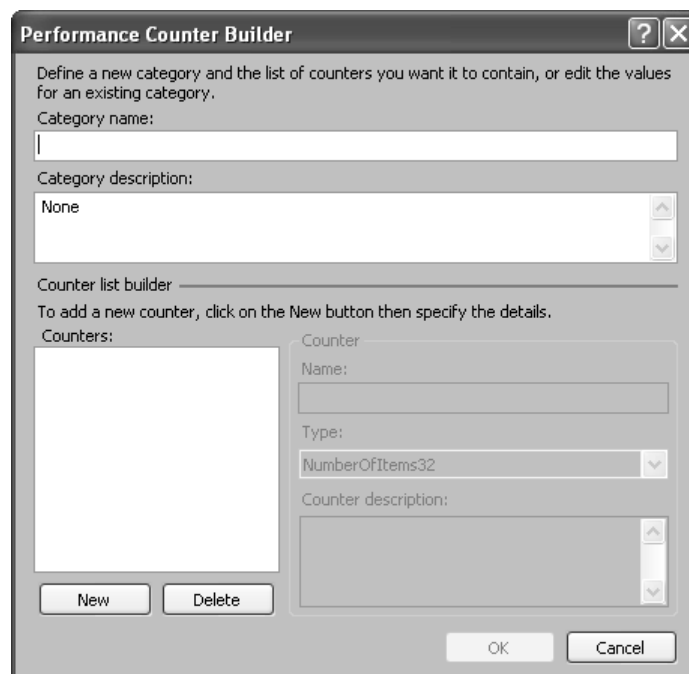


Figure 7-8 The *Performance Counter Builder* dialog box

ServiceController Component

The **ServiceController** component is used to connect to windows services and it also allows you to control the behavior of the existing services. Additionally, you can run or stop a service, manipulate a service or get information about a specific service. To add the **ServiceController** component to your form, double-click on it in the toolbar. Alternatively,

drag and drop it on the form; it will be added to the component tray. You can view its properties in the **Properties** window.

Process Component

The **Process** component is used to provide access to the processes that are running on computer. The **Process** component can be used to access the processes on both the remote and the local computers. On local computers, you can control most of the functions such as starting and terminating a process, querying it for any specific information, and so on. But on the remote computers, you cannot start and stop a process. You can only query it for any specific information. For starting a process on a local computer, you can use the following source code:

```
Private Sub Button1_Click(ByVal sender As System.Object,  
    ByVal e As System.EventArgs) Handles Button1.Click  
    Dim Process As New Process()  
    Process.StartInfo.FileName = "D:\process.txt"  
    Process.StartInfo.WindowStyle = ProcessWindowStyle.Maximized  
    Process.Start()  
End Sub
```

In this source code, you are coding a **Button** control to start a process. Here, the process name is **info_text**. For starting any process, you need to specify its full path. The **Start()** method starts the process. In this case, when you choose **Button1** on the form, the text file **info_text** will be displayed.

Timer Component

The **Timer** component sets the time interval for executing an event. This control helps you in executing the event regularly after a given time interval.

Properties of the Timer Component

There are two important properties of the **Timer** control, which are as follows:

Enabled

The **Enabled** property of the **Timer** control can be used to turn the timer on and off.

Interval

It sets the time interval between the events. The time is set in milliseconds.

The following application illustrates the use of the **BackgroundWorker** Component:

Application 1

Create an application that will start a process and show its progress with the help of the **BackgroundWorker** component.

The following steps are required to create this application:

Step 1

Start a new project and save it as **c07_VB_NET_2008_01**.

Step 2

Add two **Button** controls, a **Label** control, a **ProgressBar** control, a **ListView** control, and a **BackgroundWorker** component to the form.

Step 3

Change the **Text** property of the controls as follows:

Form1 to **BackgroundWorker**

Button1 to **Start**

Button2 to **Cancel**

Label1 to **Progress.....**

The design mode of the form will be similar to Figure 7-9.

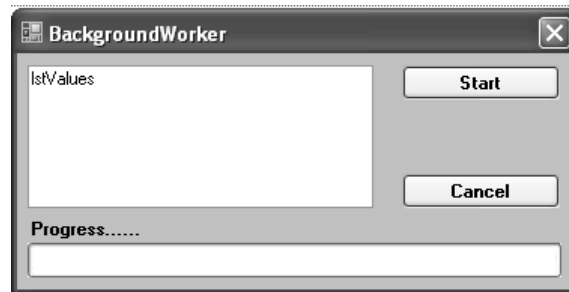


Figure 7-9 The design mode of Application 1

Step 4

Change the **Name** property of the controls as follows:

ListView to **lstValues**

Button1 to **btnStart**

Button2 to **btnCancel**

ProgressBar1 to **prgThread**

BackgroundWorker1 to **TestWorker**

Step 5

Double-click on the form and enter the following source code in the code window. The line numbers on the right are not a part of the program and are for reference only.

Option Strict Off	1
Public Class Form1	2
Public Glb_Var As Integer	3
Public Glb_Str As String	4

```

Private Sub btnStart_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles btnStart.Click           5
    btnStart.Enabled = False                                   6
    btnCancel.Enabled = True                                   7
    lstValues.Items.Clear()                                    8
    prgThread.Value = 0                                       9
    TestWorker = New System.ComponentModel.BackgroundWorker    10
    TestWorker.WorkerReportsProgress = True                   11
    TestWorker.WorkerSupportsCancellation = True              12
    TestWorker.RunWorkerAsync()                                13
End Sub                                                         14

Private Sub TestWorker_DoWork(ByVal sender As Object, _
ByVal e As System.ComponentModel.DoWorkEventArgs) _           15
Handles TestWorker.DoWork                                     16
    Dim ListText As String                                     17
    For Value As Integer = 0 To 100                           18
    If TestWorker.CancellationPending Then                     19
        Exit For                                              20
    End If                                                      21
    ListText = String.Concat("CADCIM ", Value)                22
    TestWorker.ReportProgress(Value, ListText)                 23
    Threading.Thread.Sleep(100)                                24
Next                                                            25
End Sub                                                         26

Private Sub TestWorker_ProgressChanged(ByVal sender As Object, _
ByVal e As System.ComponentModel.ProgressChangedEventArgs) _  27
Handles TestWorker.ProgressChanged                             28
    prgThread.Value = e.ProgressPercentage                     29
    lstValues.Items.Add(e.UserState)                           30
End Sub                                                         31

Private Sub TestWorker_RunWorkerCompleted(ByVal sender As Object, _
ByVal e As System.ComponentModel.RunWorkerCompletedEventArgs) _ 32
Handles TestWorker.RunWorkerCompleted                         33
    btnStart.Enabled = True                                    34
    btnCancel.Enabled = False                                  35
End Sub                                                         36

Private Sub btnCancel_Click(ByVal sender As Object, _
ByVal e As System.EventArgs) Handles btnCancel.Click          37
    TestWorker.CancelAsync()                                   38
End Sub                                                         39
End Class                                                        40

```

Explanation

The line-by-line explanation of the above given source code is as follows:

Line 1

Option Strict Off

In this line, the **Option Strict** is set to **Off**. So, all the variables should be declared before using them.

Line 2

Public Class Form1

This is the declaration of the form.

Line 3

Public Glb_Var As Integer

In this line, the **Integer** variable **Glb_Var** is declared as a global variable.

Line 4

Public Glb_Str As String

In this line, the **String** variable **Glb_Str** is declared as a global variable.

Line 5

Private Sub btnStart_Click(ByVal sender As System.Object, _ ByVal e As System.EventArgs) Handles btnStart.Click

In this line, the coding for the **Click** event of the **Start** button is done.

Lines 6-14

```
btnStart.Enabled = False  
btnCancel.Enabled = True  
lstValues.Items.Clear()  
prgThread.Value = 0  
TestWorker = New System.ComponentModel.BackgroundWorker  
TestWorker.WorkerReportsProgress = True  
TestWorker.WorkerSupportsCancellation = True  
TestWorker.RunWorkerAsync()  
End Sub
```

In these lines, when the user chooses the **Start** button, the process will start and the **Start** button will be disabled, while the **Cancel** button will be enabled. At the starting point, the **ListView** control will be blank and the value of the **ProgressBar** control will also be set to zero. The **WorkerReportProgress** property and the **WorkerSupportsCancellation** property of the **BackgroundWorker** component are set to **True**. The **RunWorkerAsync** property will start the execution of the background operation. The **End Sub** indicates the end of the procedure.

Line 15

```
Private Sub TestWorker_DoWork(ByVal sender As Object, _  
ByVal e As System.ComponentModel.DoWorkEventArgs) _  
Handles TestWorker.DoWork
```

This line refers to the **DoWork** event of the **BackgroundWorker** component.

Lines 16-25

```
Dim ListText As String
For Value As Integer = 0 To 100
If TestWorker.CancellationPending Then
Exit For
End If
ListText = String.Concat("CADCIM ", Value)
TestWorker.ReportProgress(Value, ListText)
Threading.Thread.Sleep(100)
Next
End Sub
```

In these lines, the variable **ListText** is declared as a string data type. Another variable **Value** is declared as an integer data type and the value 0 to 100 is assigned to it. This code will basically concatenate the values of the **ListText** and **Value** variables and display them in the **ListView** control. The **ReportProgress** method of the **BackgroundWorker** component raises the **ProgressChanged** event of the **BackgroundWorker** component. The **Threading** namespace contains the **Thread** class and the **Sleep** is the method of the **Thread** class. The **Sleep** method is used to suspend the current thread for a specific period of time.

Lines 26-29

```
Private Sub TestWorker_ProgressChanged(ByVal sender As Object, _
ByVal e As System.ComponentModel.ProgressChangedEventArgs) _
Handles TestWorker.ProgressChanged
prgThread.Value = e.ProgressPercentage
lstValues.Items.Add(e.UserState)
End Sub
```

These lines refer to the **ProgressChanged** event of the **BackgroundWorker** component. The **e.ProgressPercentage** will return the integer value, which is assigned to the **Value** property of the progress bar **PrgThread**. This value will be added to the listview **lstValues**.

Line 30

```
Private Sub TestWorker_RunWorkerCompleted(ByVal sender As Object, _
ByVal e As System.ComponentModel.RunWorkerCompletedEventArgs) _
Handles TestWorker.RunWorkerCompleted
```

This line refers to the **RunWorkerCompleted** event of the **BackgroundWorker** component.

Lines 31-33

```
btnStart.Enabled = True
btnCancel.Enabled = False
End Sub
```

In these lines, the **Start** button is enabled and the **Cancel** button is disabled.

Lines 34-35

```
Private Sub btnCancel_Click(ByVal sender As Object, _
ByVal e As System.EventArgs) Handles btnCancel.Click
```

TestWorker.CancelAsync()

These lines refer to the **Click** event of the **Cancel** button. The **CancelAsync** method of the **BackgroundWorker** component is used to cancel the pending background operations.

Line 35

End Sub

This line indicates the end of the procedure.

Line 36

End Class

This line indicates the end of the class **Form1**.

Step 6

Execute the application; the output form will be displayed. Choose the **Start** button on the form; the output will be displayed, as shown in Figure 7-10.

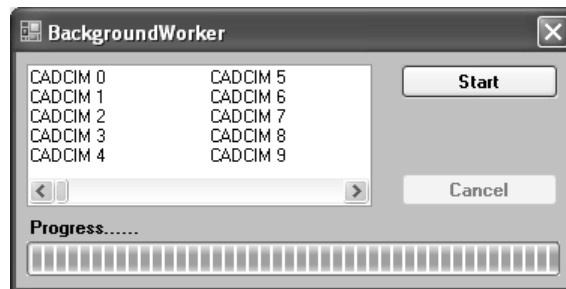


Figure 7-10 The output of Application 1

The following application illustrates use of the **ErrorProvider** and **ImageList** components:

Application 2

Create an application that will prompt the user to enter his personal details in a form and then submit them.

In this application, the user will be prompted to enter his personal details in a form. In case, the user leaves any of the fields blank, an error icon will be displayed.

The following steps are required to create this application:

Step 1

Start a new project and save it as **c07_VB_NET_2008_02**.

Step 2

Add six **Label** controls, six **TextBox** controls, three **Button** controls, a **ListView** control, and a **GroupBox** control to the form. Also, add an **ErrorProvider** component and an **ImageList** component that will be added to the component tray.

The design mode of the application will be similar to Figure 7-11.



Figure 7-11 The design mode of Application 2

Step 3

Change the **Name** property of the controls as follows:

TextBox1 to **Txt_Name**
TextBox2 to **Txt_Address**
TextBox3 to **Txt_Phone**
TextBox4 to **Txt_Designation**
TextBox5 to **Txt_JDate**
TextBox6 to **Txt_image**
Button1 to **Btn_upload**
Button2 to **Btn_Submit**
Button3 to **Btn_Exit**
GroupBox1 to **Grp_image**
ListView1 to **LstView_Image**

Step 4

Double-click on the form and enter the following source code in the code window:

```
Dim Bol As Boolean                                1
Private Sub Form1_Load(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles MyBase.Load    2
    Grp_Image.Visible = False                        3
    Bol = False                                      4
End Sub                                              5
```


Private Function ValidateTextBox(ByVal Txt As TextBox) As Boolean	6
If Txt.Text = "" Then	7
Me.ErrorProvider1.SetError(Txt, "Please enter the text")	8
Txt.Focus()	9
End If	10
End Function	11
Private Sub Btn_upload_Click(ByVal sender As System.Object, _ ByVal e As System.EventArgs) Handles Btn_upload.Click	12
Grp_Image.Visible = True	13
End Sub	14
Private Sub Btn_Exit_Click(ByVal sender As System.Object, _ ByVal e As System.EventArgs) Handles Btn_Exit.Click	15
End	16
End Sub	17
Private Sub LstView_Image_DoubleClick(ByVal sender As Object, _ ByVal e As System.EventArgs) Handles LstView_Image.DoubleClick	18
Dim str As String	19
Dim indexs As Integer = (LstView_Image.SelectedItems(0).Index)	20
str = LstView_Image.Items(indexs).Text	21
Txt_image.Text = str	22
If Txt_image.Text = "" Then	23
Me.ErrorProvider1.SetError(Txt_image, _ "Please select the image from the Listview")	24
Exit Sub	25
End If	26
Grp_Image.Visible = False	27
End Sub	28
Private Sub Txt_Address_GotFocus(ByVal sender As Object, _ ByVal e As System.EventArgs) Handles Txt_Address.GotFocus	29
ValidateTextBox(Txt_Name)	30
Text_Checked(Txt_Name)	31
End Sub	32
Private Sub Txt_JDate_GotFocus(ByVal sender As Object, _ ByVal e As System.EventArgs) Handles Txt_JDate.GotFocus	33
ValidateTextBox(Txt_Desgination)	34
Text_Checked(Txt_Desgination)	35
End Sub	36
Private Sub Txt_Phone_GotFocus(ByVal sender As Object, _ ByVal e As System.EventArgs) Handles Txt_Phone.GotFocus	37
ValidateTextBox(Txt_Address)	38
Text_Checked(Txt_Address)	39
End Sub	40

```

Private Sub Txt_image_GotFocus(ByVal sender As Object, _
ByVal e As System.EventArgs) Handles Txt_image.GotFocus           41
ValidateTextBox(Txt_JDate)                                         42
Text_Checked(Txt_JDate)                                           43
End Sub                                                            44

Private Sub Txt_image_KeyPress(ByVal sender As Object, _
ByVal e As System.Windows.Forms.KeyPressEventArgs) _
Handles Txt_image.KeyPress                                         45
Me.Btn_upload.Focus()                                             46
End Sub                                                            47

Private Sub Btn_Submit_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Btn_Submit.Click             48
If Txt_image.Text = "" Then                                       49
Me.ErrorProvider1.SetError(Txt_image, _
"Please select the image from the Listview")                       50
Exit Sub                                                         51
End If                                                            52
MsgBox("Thanks For The Registration")                             53
End Sub                                                            54
Private Sub Text_Checked(ByVal txt As TextBox)                     55
If Not txt.Text = "" Then                                         56
Me.ErrorProvider1.Clear()                                         57
End If                                                            58
End Sub                                                            59

```

Explanation

The line-by-line explanation of the above given source code is as follows:

Lines 6-11

Private Function ValidateTextBox(ByVal Txt As TextBox) As Boolean

If Txt.Text = "" Then

Me.ErrorProvider1.SetError(Txt, "Please enter the text")

Txt.Focus()

End If

End Function

In these lines, the function **ValidateTextBox** is declared as **Boolean** type. If an empty string is assigned to the **Text** property of the **TextBox** control, an error icon will be displayed because of the **ErrorProvider** component. If you move the cursor over the icon, the message **Please enter the text** will be displayed.

Lines 18-28

Private Sub LstView_Image_DoubleClick(ByVal sender As Object, _

ByVal e As System.EventArgs) Handles LstView_Image.DoubleClick

Dim str As String

Dim indexs As Integer = (LstView_Image.SelectedItems(0).Index)

```

str = LstView_Image.Items(indexs).Text
Txt_image.Text = str
If Txt_image.Text = "" Then
Me.ErrorProvider1.SetError(Txt_image,
    "Please select the image from the Listview")
Exit Sub
End If
Grp_Image.Visible = False
End Sub

```

In these lines, the **Double-Click** event of the **ListView** control is coded. It means whenever you double-click on an image from the **ListView** control, the image name will be added to the **TextBox** control associated with it. If you do not select any image, then the **ErrorProvider** component will give an error message **Please select the image from the Listview**.

Lines 29-32

```

Private Sub Txt_Address_GotFocus(ByVal sender As Object, _
ByVal e As System.EventArgs) Handles Txt_Address.GotFocus
ValidateTextBox(Txt_Name)
Text_Checked(Txt_Name)
End Sub

```

These lines refer to the **GotFocus** event of the **Txt_Address** textbox. Here, the **Txt_Name** textbox will be validated and checked.

Step 5

Execute the application; the output form will be displayed, as shown in Figure 7-12. Enter the values in all the textboxes and then choose the **Submit** button; a message box with the message **Thanks For The Registration** will be displayed, as shown in Figure 7-13.

Figure 7-12 The output form of Application 2

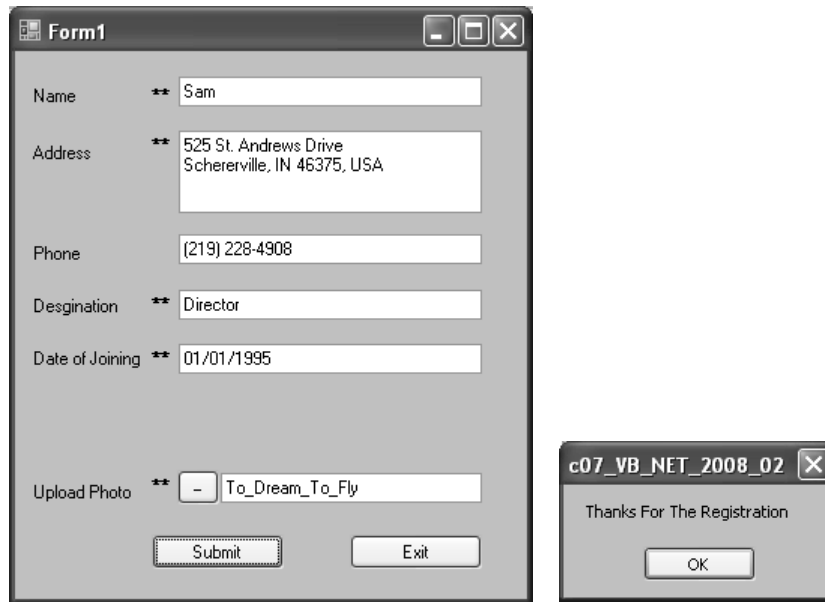


Figure 7-13 Message displayed after entering the values

The following application illustrates the use of the **FileSystemWatcher** component:

Application 3

Create an application that will use the **FileSystemWatcher** component to keep a watch on a particular directory.

The following steps are required to create this application:

Step 1

Start a new project and save it as **c07_VB_NET_2008_03**.

Step 2

Add a **FileSystemWatcher** component to the form.

Step 3

Double-click on the form and enter the following source code in the code window:

```
Private Sub Form1_Load(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles Me.Load           1
    With FileSystemWatcher1                                  2
        .Path = My.Computer.FileSystem.CurrentDirectory & "\FileSystemWatcher" 3
        .EnableRaisingEvents = True                        4
        .IncludeSubdirectories = False                       5
        .Filter = "TxtFilesystem.txt"                      6
    End With
End Sub
```

End With	7
End Sub	8
Private Sub FileSystemWatcher1_Created(ByVal sender As Object, _ ByVal e As System.IO.FileSystemEventArgs) _ Handles FileSystemWatcher1.Created	9
MsgBox("New Text file" & e.Name & " has been created")	10
End Sub	11
Private Sub FileSystemWatcher1_Deleted(ByVal sender As Object, _ ByVal e As System.IO.FileSystemEventArgs) _ Handles FileSystemWatcher1.Deleted	12
MsgBox("Text file" & e.Name & " has been deleted")	13
End Sub	14

Explanation

In the above source code, the **Load** event of the form is coded first. Then, the **With-End With** statements are used for assigning the values to different properties of the **FileSystemWatcher** component. The **FileSystemWatcher** component will keep a watch on the current directory of the file system. Therefore, whenever a new file is created in the particular directory, the application will give the message **Text file has been created** and whenever an existing file has been deleted, the message **Text file has been deleted** will be displayed.

The following application illustrates the use of the **Process** component:

Application 4

Create an application that will search and display the current processes and applications. It should also display the list of visible applications, processes, and ids.

This application will prompt the user to enter the name of a process or an application to check whether the process or the application exists or not. The application will also enable the user to view the list of visible applications, processes, and ids.

The following steps are required to create this application:

Step 1

Start a new project and save it as **c07_VB_NET_2008_04**.

Step 2

Add three **GroupBox** controls, five **Button** controls, and three **TextBox** controls. The resultant form will appear similar to Figure 7-14.

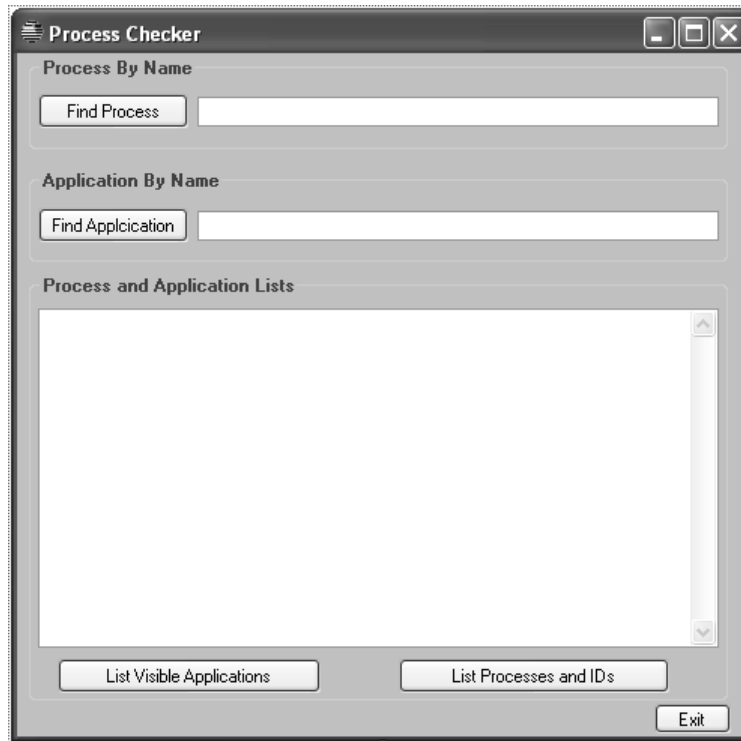


Figure 7-14 The design mode of Application 4

Step 3

Go to **Solution Explorer** and double-click on the **My Project** option; a window with different options will be displayed, as shown in Figure 7-15.

Step 4

Select the **References** option from the window and then choose the **Add** button; the **Add Reference** dialog box will be displayed, as shown in Figure 7-16. Now, go back to the design window.

Step 5

Change the **Name** property of the controls as follows:

Form1 to **Frm_ProcessChk**

Button1 to **Btn_Proc**

Button2 to **Btn_App**

Button3 to **Btn_Application**

Button4 to **Btn_Process**

Button5 to **Btn_Exit**

TextBox1 to **Txt_Process**

TextBox2 to **Txt_Application**

TextBox3 to **Txt_ListProcess**

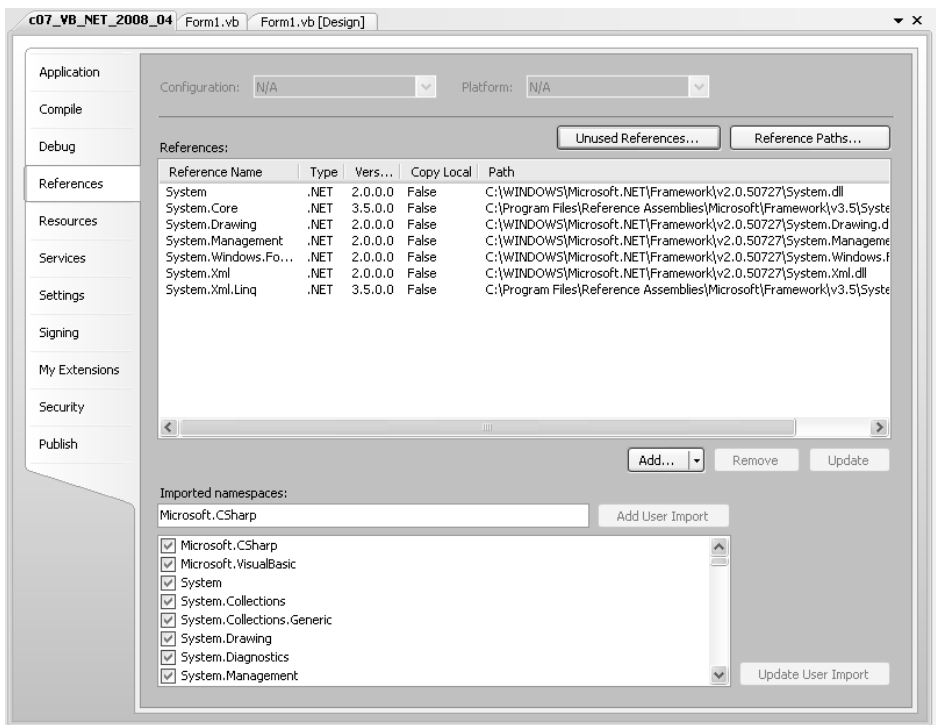


Figure 7-15 The window displayed on double-clicking on the MyProject option

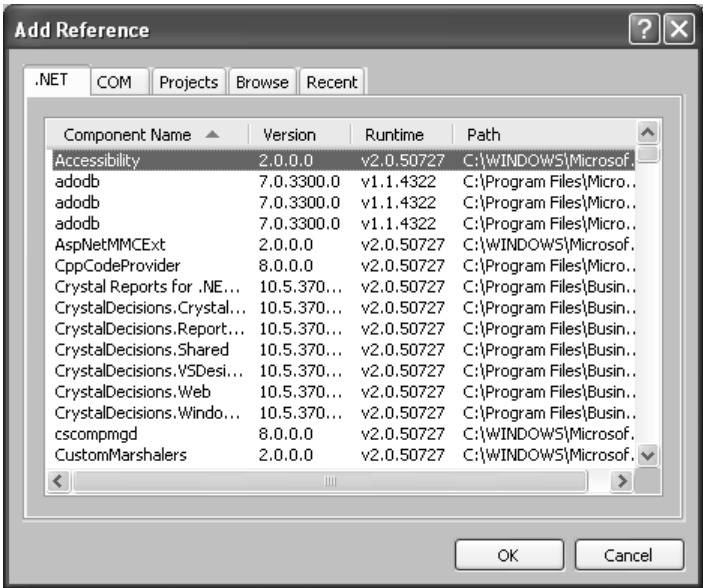


Figure 7-16 The Add Reference dialog box

Step 6

Double-click on the form and enter the following source code in the code window:

```
Imports System.Text                                1
Imports System.Management                          2
Public Class Frm_ProcessChk                          3

    Private Sub Btn_Application_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Btn_Application.Click      4
        Dim StrBlder As New StringBuilder()              5
        Dim pProc As New Process()                      6

        For Each pProc In Process.GetProcesses(".")      7
            If pProc.MainWindowTitle.Length > 0 Then
                StrBlder.Append("Window Title: " + _
                pProc.MainWindowTitle.ToString() + Environment.NewLine)  8
                StrBlder.Append("Process Name: " + _
                pProc.ProcessName.ToString() + Environment.NewLine)      9
                StrBlder.Append("Window Handle: " + _
                pProc.MainWindowHandle.ToString() + Environment.NewLine) 10
                StrBlder.Append("Memory Allocation: " + _
                pProc.PrivateMemorySize64.ToString() + Environment.NewLine) 11
                StrBlder.Append(Environment.NewLine)      12
            End If                                       13
        Next                                           14
        Txt_ListProcess.Text = StrBlder.ToString      15
    End Sub                                           16

    Private Sub Btn_Exit_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Btn_Exit.Click      17
    End Sub                                           18
End Sub                                           19

    Private Sub Btn_Process_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Btn_Process.Click      20
        Dim Str_Bld As New StringBuilder()              21
        Dim MgtClass As New ManagementClass("Win32_Process")    22
        Dim MgtObj As New ManagementObject()              23
        For Each MgtObj In MgtClass.GetInstances            24
            Str_Bld.Append("Name: " & MgtObj("Name") & Environment.NewLine) 25
            Str_Bld.Append("ID: " & MgtObj("ProcessId") & Environment.NewLine) 26
            Str_Bld.Append(Environment.NewLine)            27
        Next                                               28
        Txt_ListProcess.Text = Str_Bld.ToString            29
    End Sub                                           30
```



```

Private Sub Btn_App_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles Btn_App.Click           31
Dim Bol_Proc As Boolean = False                                32
Dim Proc As New Process()                                     33
Dim AppName As String = Trim(Txt_Application.Text)            34
For Each Proc In Process.GetProcesses(".")                     35
If Proc.ProcessName.ToLower() = AppName.ToLower() Then       36
Bol_Proc = True                                                37
End If                                                         38
Next                                                            39
If Bol_Proc = True Then                                        40
MessageBox.Show(Txt_Application.Text + " process name found.", _
"CADCIM Technologies")                                         41
Else                                                            42
MessageBox.Show(Txt_Application.Text + " process name not found.", _
"CADCIM Technologies")                                         43
End If                                                         44
End Sub                                                         45

Private Sub Btn_Proc_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Btn_Proc.Click           46
Dim MgtClass As New ManagementClass("Win32_Process")          47
Dim MgtObj As New ManagementObject()                          48
Dim Bol_Proc As Boolean = False                                49
Dim Str_Proc As String = Trim(Txt_Process.Text)                50
For Each MgtObj In MgtClass.GetInstances()                     51
If MgtObj("Name").ToString().ToLower() = Str_Proc.ToLower() Then 52
Bol_Proc = True                                                53
End If                                                         54
Next                                                            55
If Bol_Proc = True Then                                        56
MessageBox.Show(Txt_Process.Text & " process name found.", _
"CADCIM Technologies")                                         57
Else                                                            58
MessageBox.Show(Txt_Process.Text & " process name not found.", _
"CADCIM Technologies")                                         59
End If                                                         60
End Sub                                                         61
End Class                                                       62

```

Explanation

The line-by-line explanation of the above given source code is given as follows:

Lines 1-2

Imports System.Text

Imports System.Management

In these lines, the **Text** and **Management** namespaces are imported.

Line 3

Public Class Frm_ProcessChk

In this line, the class **Frm_ProcessChk** is declared.

Lines 7-15

```
For Each pProc In Process.GetProcesses(".")
If pProc.MainWindowTitle.Length > 0 Then
  StrBlder.Append("Window Title: " + pProc.MainWindowTitle.ToString() + _
  Environment.NewLine)
  StrBlder.Append("Process Name: " + pProc.ProcessName.ToString() + _
  Environment.NewLine)
  StrBlder.Append("Window Handle: " + pProc.MainWindowHandle.ToString() + _
  Environment.NewLine)
  StrBlder.Append("Memory Allocation: " + pProc.PrivateMemorySize64.ToString() + _
  Environment.NewLine)
  StrBlder.Append(Environment.NewLine)
End If
Next
```

Txt_ListProcess.Text = StrBlder.ToString

In these lines, the **For** loop is used to check all the processes in the system. Whenever a user chooses the **ListAvailableApplications** button, all the applications running on the system will be listed in the multiline **TextBox** control and the information about **Window Title**, **Process Name**, **Window Handle**, and **Memory Allocation** will be displayed.

Lines 20-29

```
Private Sub Btn_Process_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles Btn_Process.Click
  Dim Str_Bld As New StringBuilder()
  Dim MgtClass As New ManagementClass("Win32_Process")
  Dim MgtObj As New ManagementObject()
  For Each MgtObj In MgtClass.GetInstances
    Str_Bld.Append("Name: " & MgtObj("Name") & Environment.NewLine)
    Str_Bld.Append("ID: " & MgtObj("ProcessId") & Environment.NewLine)
    Str_Bld.Append(Environment.NewLine)
  Next
```

Txt_ListProcess.Text = Str_Bld.ToString

In these lines, the variable **MgtClass** is declared as the **ManagementClass** type. The variable **MgtObj** is declared as an instance of the **ManagementObject** class. The **For Each** loop is used to check the processes of the system. Whenever a user chooses the **ListProcesses and IDs** button, the **For** loop starts searching for all the executable processes on the system and all of them will be displayed in the multiline **TextBox** control.

Step 7

Execute the application; the output of the application will be displayed. Enter any process that you want to search, for example, **smss.exe**, in the **Find Process** textbox. Next, choose the **Find Process** button; a message box with the message **smss.exe process name found**. will be displayed, as shown in Figure 7-17. Similarly, if you want to search any application,

then enter the name of the application in the textbox on the right of the **Find Application** button, and then choose the **Find Application** button. Next, if you want to view all the currently visible applications, the choose **List Visible Applications** button; all the visible applications will be listed in the textbox, as shown in Figure 7-18. Similarly, to view all the processes, choose the **List Processes and IDs** button, all the processes will be listed in the textbox, as shown in Figure 7-19.

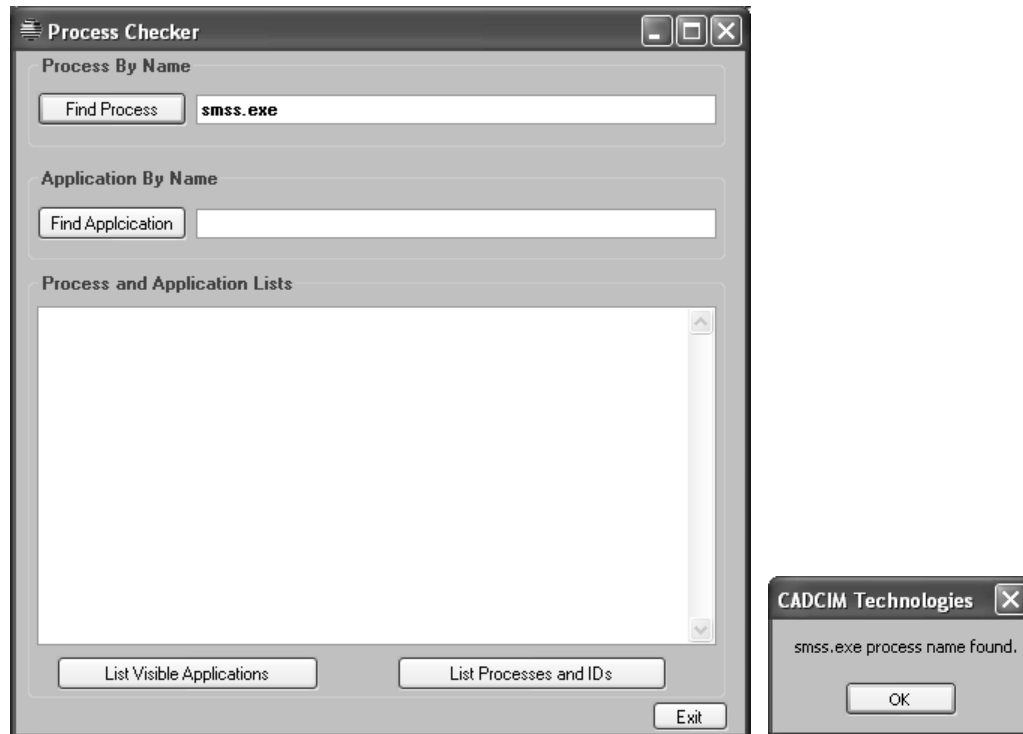


Figure 7-17 The output of the Application 4 in case of finding a process

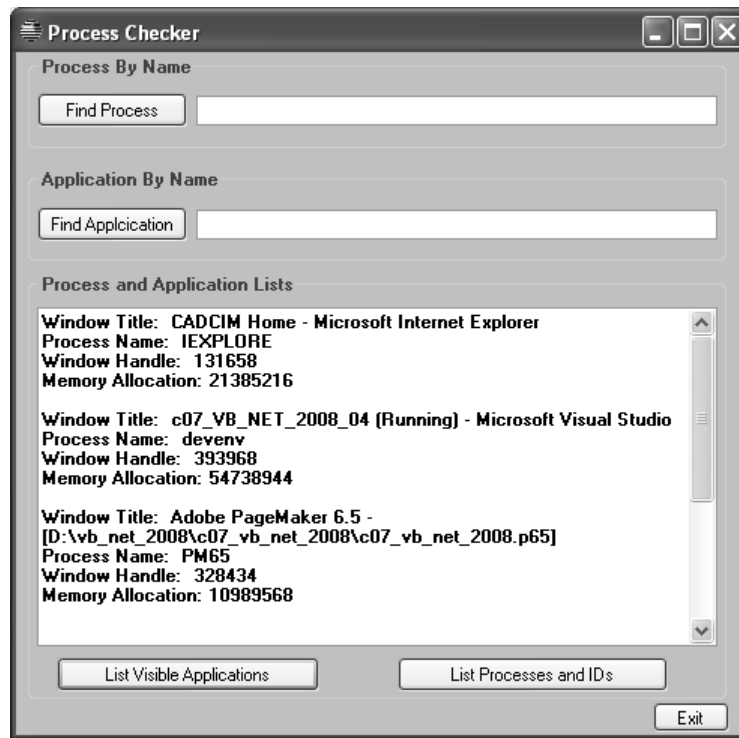


Figure 7-18 All the applications listed in the textbox

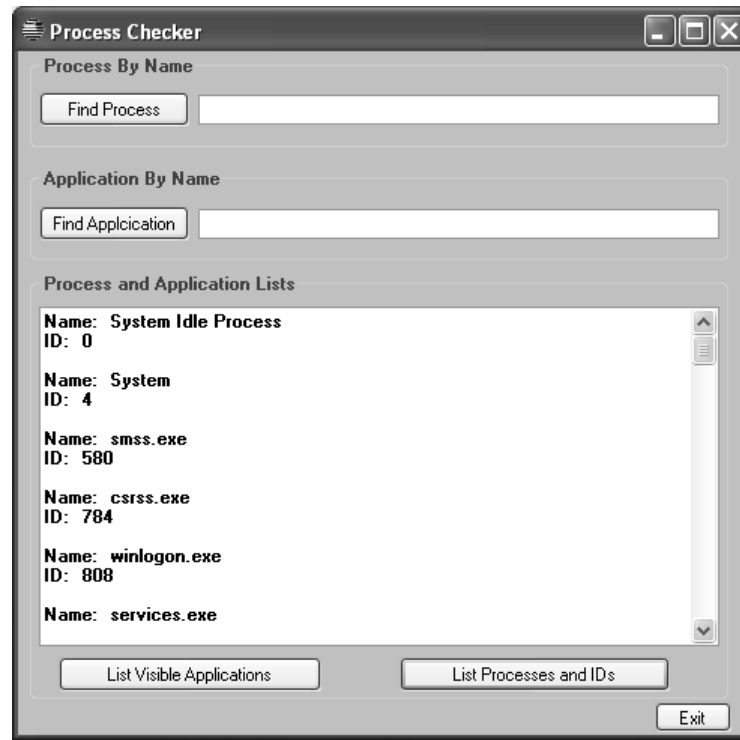


Figure 7-19 All the applications listed in textbox

The following application illustrates the use of the **ServiceController** component:

Application 5

Create an application that will start and stop a process.

The following application will start and stop a process. If you enter the process that does not exist in the system, the application will give an error message.

The following steps are required to create this application:

Step 1

Start an application and save it as **c07_VB_NET_2008_05**.

Step 2

Add a **Label** control, a **TextBox** control, a **ServiceController** component, and two **Button** controls to the form.

The resultant form will appear as shown in Figure 7-20.



Figure 7-20 The design mode of Application 5

Step 3

Change the **Name** property of the controls as follows:

TextBox1 to **Txt_SerName**

ServiceController1 to **ServiceContr**

Step 4

Double-click on the form and enter the following source code:

```
Imports System.Collections.Generic           1
Imports System.ComponentModel             2
Imports System.ServiceProcess            3
Public Class Form1                       4
    Private Sub Form1_Load(ByVal sender As Object, _
        ByVal e As System.EventArgs) Handles Me.Load           5
        Button2.Enabled = False                               6
        Txt_SerName.Focus()                                   7
    End Sub                                                    8
    Private Sub Button1_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles Button1.Click      9
        Try                                                    10
            If Trim(Txt_SerName.Text) = "" Then
                MessageBox.Show("Enter the Service Name", "CADCIM Technologies") 11
            Exit Sub                                           12
            ServiceContr = New ServiceController(Trim(Txt_SerName.Text)) 13
            If ServiceContr.Status = ServiceControllerStatus.Running Then 14
                MessageBox.Show("Service already running.....", "CADCIM Technologies", _ 15
                    MessageBoxButtons.OK, MessageBoxIcon.Information)
                Button2.Enabled = True                           16
                Button1.Enabled = False                           17
            Exit Sub                                           18
            End If                                             19
            If Not Trim(Txt_SerName.Text) = "" Then            20
                ServiceContr.Start()                             21
                Button2.Enabled = True                           22
            End If
        End Try
    End Sub
End Class
```

Button1.Enabled = False	23
Label1.Text = Txt_SerName.Text & " " & "Service started....."	24
Else	25
End If	26
Catch ex As Exception	27
MsgBox(ex.Message)	28
End Try	29
End Sub	30
Private Sub Button2_Click(ByVal sender As System.Object, _ ByVal e As System.EventArgs) Handles Button2.Click	31
ServiceContr = New ServiceController(Trim(Txt_SerName.Text))	32
If ServiceContr.Status = ServiceControllerStatus.Stopped Then	33
Button2.Enabled = False	34
Button1.Enabled = True	35
MessageBox.Show("Service already stopped.....", "CADCIM Technologies", _ MessageBoxButtons.OK, MessageBoxIcon.Information)	36
Exit Sub	37
End If	38
If ServiceContr.CanStop Then	39
ServiceContr.Stop()	40
Label1.Text = Txt_SerName.Text & " " & "Service stopped....."	41
Button2.Enabled = False	42
Button1.Enabled = True	43
End If	44
End Sub	45
End Class	46

Explanation

The line-by-line explanation of the above given source code is as follows:

Lines 1-3

Imports System.Collections.Generic

Imports System.ComponentModel

Imports System.ServiceProcess

In these lines, the namespaces **System.Collections.Generic**, **System.ComponentModel**, **System.ServiceProcess** are imported.

Lines 9-30

Private Sub Button1_Click(ByVal sender As System.Object, _

ByVal e As System.EventArgs) Handles Button1.Click

Try

If Trim(Txt_SerName.Text) = "" Then

MessageBox.Show("Enter the Service Name", "CADCIM Technologies")

Exit Sub

ServiceContr = New ServiceController(Trim(Txt_SerName.Text))

If ServiceContr.Status = ServiceControllerStatus.Running Then

```

MessageBox.Show("Service already running.....", "CADCIM Technologies", _
MessageBoxButtons.OK, MessageBoxIcon.Information)
Button2.Enabled = True
Button1.Enabled = False
Exit Sub
Label1.Text = Txt_SerName.Text & " " & "Service started....."
Else
End If
Catch ex As Exception
MsgBox(ex.Message)
End Try
End Sub

```

In these lines, the **Try-End Try** statement is used to handle exceptions, if there are any exceptions. If a null value is assigned to the **TextBox** control, a message box with the message **Enter the Service Name** will be displayed. In the line **ServiceContr = New ServiceController(Trim(Txt_SerName.Text))**, any value that the user enters in the **TextBox** control will be assigned to the **ServerController** component. If the service name that you have entered is already running, the message box with the message **Service already running.....** will be displayed. In this case, **Button2** will be enabled and **Button1** will be disabled. The service name with the message **Service started.....** will be displayed in the **Label** control. The **Catch** statement will catch the system exception and display a message. Similarly, the service control will stop when the user chooses the **Button2** control.

Step 5

Execute the application; the output form will be displayed. Enter any service in the **Server Name** textbox such as **FTP Publishing** and choose the **Start Service** button; the service will started. Check the service in the **Computer Management** window. To check the service, double-click on **My Computer**; the **Computer Management** window will be displayed. Next, expand the **Services and Applications** nodes and then double-click on the **Services** node; all the services will be displayed. In this case, status of the service is **Started**, as shown in Figure 7-21.

Step 6

Choose the **Stop Service** button; the service will be stopped. Check the status of the service again, the value of the status will be null, as shown in Figure 7-22.



Figure 7-21 The output of Application 5 in case of starting the service



Figure 7-22 The output of Application 5 in case of stopping the service

The following application illustrates the use of the **PerformaneCounter** and **Timer** components:

Application 6

Create an application to check whether the name of the performance counter entered by the user exists or not.

In this application, the user will be prompted to enter the **ObjectName**, **CounterName**, and **InstanceName**. The application will check whether the data entered by the user exists or not and then it will display a message.

The following steps are required to create this application:

Step 1

Start a new project and save it as **c07_VB_NET_2008_06**.

Step 2

Add three **Label** controls, three **TextBox** controls, a **PerformanceCounter** component, and two **Button** controls to the form.

The resultant form will appear as shown in Figure 7-23.

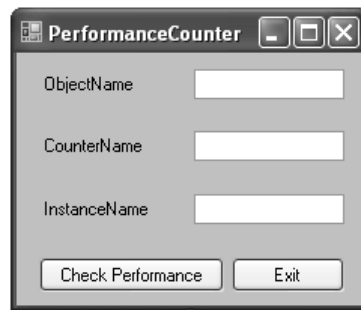


Figure 7-23 The design mode of Application 6

Step 3

Change the **Name** property of the controls as follows:

TextBox1 to **Txt_objectName**

TextBox2 to **Txt_CounterName**

TextBox3 to **Txt_InstanceName**

Step 4

Double-click on the **CheckPerformance** button. Now, declare the global variables and then enter the source code for the **Click** event of the button as follows:

```
Public Class Form1                                1
Public objectName As String                        2
Public counterName As String                      3
Public instanceName As String                    4
Public Counter As PerformanceCounter              5

Public Sub Button1_Click(ByVal sender As Object, _ 6
    ByVal e As System.EventArgs) Handles Button1.Click
If Txt_CounterName.Text = "" Or Txt_objectName.Text = "" Or _ 7
    Txt_InstanceName.Text = "" Then Exit Sub
objectName = Txt_objectName.Text                  8
counterName = Txt_CounterName.Text                9
instanceName = Txt_InstanceName.Text              10
```

```

objectName = objectName.Trim()           11
counterName = counterName.Trim()         12
instanceName = instanceName.Trim()       13
If (Not PerformanceCounterCategory.Exists(objectName)) Then 14
    MessageBox.Show("Object" & objectName & " does not exists! ", _
        "CADCIM Technologies.....")    15
Exit Sub                                 16
End If                                   17
If (Not PerformanceCounterCategory.CounterExists(counterName, _
    objectName)) Then                    18
    MessageBox.Show("Counter " & counterName & " does not exist! ", _
        "CADCIM Technologies.....")    19
Exit Sub                                 20
End If                                   21
MessageBox.Show("Object Name :" & objectName & Chr(13) & _
    "Counter Name: " & counterName & Chr(13) & "& Instance Name: " _
    & instanceName & Chr(13) & " exists", "CADCIM Technologies") 22
End Sub                                  23

Private Sub Button2_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Button2.Click        24
End                                         25
End Sub                                    26
End Class                                  27

```

Explanation

The line-by-line explanation of the above mentioned source code is as follows:

Lines 1-5

Public Class Form1

Public objectName As String

Public counterName As String

Public instanceName As String

Public theCounter As PerformanceCounter

In these lines, the class **Form1** is declared. Also, the variables are declared globally.

Lines 6-13

Public Sub Button1_Click(ByVal sender As Object, _

ByVal e As System.EventArgs) Handles Button1.Click

If Txt_CounterName.Text = "" Or Txt_objectName.Text = "" Or _
Txt_InstanceName.Text = "" Then Exit Sub

objectName = Txt_objectName.Text

counterName = Txt_CounterName.Text

instanceName = Txt_InstanceName.Text

objectName = objectName.Trim()

counterName = counterName.Trim()

instanceName = instanceName.Trim()

In these lines, the **Click** event of **Button1** is coded. If the null value is assigned to all the **TextBox** controls or any one of them, the control will exit the sub procedure. The value of the **Txt_objectName** textbox will be assigned to the variable **objectName**. The value of the **Txt_CounterName** textbox will be assigned to the variable **counterName**. The value of the **Txt_InstanceName** textbox will be assigned to the variable **instanceName**. The **objectName.Trim()** is used to trim the extra spaces, if any.

Lines 14-23

```

If (Not PerformanceCounterCategory.Exists(objectName)) Then
MessageBox.Show("Object " & objectName & " does not exist! ", _
"CADCIM Technologies.....")
Exit Sub
End If
MessageBox.Show("Object Name :" & objectName & Chr(13) & _
"Counter Name: " & counterName & Chr(13) & "& Instance Name: " _
& instanceName & Chr(13) & " exists", "CADCIM Technologies")
End Sub

```

In the above lines, if the object name entered by the user does not exist, then a message box with the message **does not exist** concatenated with the object name will be displayed. And, if the object name entered by the user exists, then a message box with the **Object Name**, **Counter Name**, and **Instance Name** will be displayed. In these lines, 13 is the ASCII value of the ENTER key.

Step 5

Execute the application; the output of the form will be displayed. Enter the values **FTP Service**, **Bytes Received/sec**, and **_Total** for the **ObjectName**, **CounterName**, and **InstanceName** parameters, respectively. Next, choose the **Check Performance** button; the output will be displayed, as shown in Figure 7-24. If you enter the invalid entries, the output will be displayed, as shown in Figure 7-25.

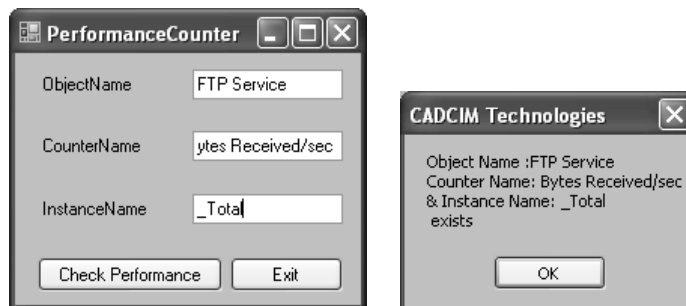


Figure 7-24 The output of Application 6 in case of valid entries

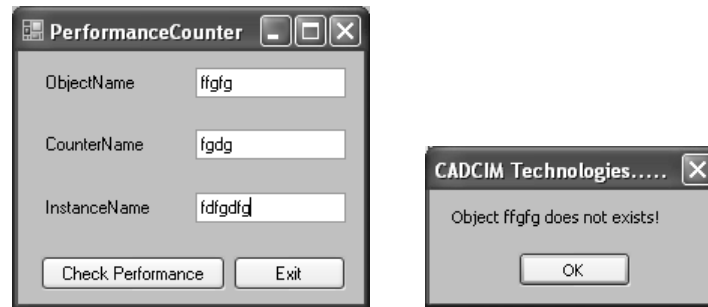


Figure 7-25 The output of Application 6 in case of invalid entries

The following application illustrates the use of the **MessageQueue** component.

Application 7

Create an application to send the messages to the local server and also to retrieve those messages.

In this application, the user will be prompted to enter a subject and message on a form. This message will then be displayed in the **Read the Message** area.

The following steps are required to create this application:

Step 1

Start a new project and save it as **c07_VB_NET_2008_07**.

Step 2

Add two **GroupBox** controls, three **TextBox** controls, one **ComboBox** control, and a **Button** control to the form.

The resultant form will appear as shown in Figure 7-26.



Figure 7-26 The design mode of Application 7

Step 3

Change the value of the **Name** property of controls as follows:

TextBox1 to **Txt_Label**
TextBox2 to **Txt_Send**
TextBox3 to **Txt_Read**
ComboBox1 to **CmbQueue**
Button1 to **Btn_Send**

Step 4

Enter the following source code in the code window:

```
Imports System.Messaging           1
Public Class Form1                 2
Dim MsgQ As MessageQueue          3
Private Sub Form1_Load(ByVal sender As Object, _
ByVal e As System.EventArgs) Handles Me.Load      4
If Not (MessageQueue.Exists(".\Private$\CADCIM")) Then 5
MsgQ = MessageQueue.Create(".\Private$\CADCIM")    6
Else                                              7
MsgQ = New MessageQueue(".\Private$\CADCIM")      8
End If                                           9
End Sub                                         10

Private Sub Btn_Send_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Btn_Send.Click 11
Dim msg As Message                          12
Try                                           13
If Txt_Label.Text = "" Then                 14
MessageBox.Show("Enter the subject", "CADCIM") 15
Exit Sub                                    16
End If                                       17
If Txt_Send.Text = "" Then                  18
MessageBox.Show("Enter the text for the message", "CADCIM") 19
Exit Sub                                    20
End If                                       21

msg = New Message                           22
msg.Priority = MessagePriority.Normal        23
msg.Label = Txt_Label.Text                  24
msg.Body = Txt_Send.Text                    25
MsgQ.Send(msg)                              26
MessageBox.Show("Message sent.")            27
For Each msg In MsgQ                        28
CmbQueue.Items.Add(msg.Label)               29
Next                                         30
Catch ex As Exception                       31
```

MessageBox.Show("Error: " + ex.ToString())	32
End Try	33
End Sub	34
Private Sub Btn_Read_Click(ByVal sender As System.Object, _ ByVal e As System.EventArgs)	35
End Sub	36
Private Sub CmbQueue_SelectedIndexChanged(ByVal sender As System.Object, _ ByVal e As System.EventArgs) Handles CmbQueue.SelectedIndexChanged	37
Dim str() As String = {"System.String,mscorlib"}	38
MsgQ.Formatter = New XmlMessageFormatter(str)	39
Dim Mymessage As Message = MsgQ.Receive()	40
Txt_Read.Text = Mymessage.Body	41
End Sub	42
End Class	43

Explanation

The line-by-line explanation of the above given source code is as follows:

Line 1

Imports System.Messaging

In this line, the namespace **System.Messaging** is imported.

Line 2

Public Class Form1

In this line, the class **Form1** is declared.

Line 3

Dim MsgQ As MessageQueue

In this line, the variable **MsgQ** is declared as **MessageQueue** that will provide the access to queue on the message queuing server.

Line 4

Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load

In this line, the form load event is declared.

Line 5

If Not (MessageQueue.Exists(".\Private\$\CADCIM")) Then

This line will determine whether the message queuing exists on the specified path or not. If it exists, then the control will be transferred to the next line.

Line 6

MsgQ = MessageQueue.Create(".\Private\$\CADCIM")

This line will create a non-transactional message queuing queue at the specified path.

Lines 7-9

Else

MsgQ = New MessageQueue(".\Private\$\CADCIM")

End If

If the condition in the previous line does not satisfy, the control will be transferred to line 7. In this line, the keyword **New** is used to create a new object instance of **MessageQueue**. **EndIf** indicates the end of the **If** statement.

Line 11

**Private Sub Btn_Send_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Btn_Send.Click**

In this line, the **Click** event of the **Button** control is declared.

Lines 13-17

Try

If Txt_Label.Text = "" Then

MessageBox.Show("Enter the Subject", "CADCIM")

Exit Sub

End If

In these lines, the **Try-Catch** block is used to catch the exceptions, if any. If the null string is assigned to the **Txt_Label** textbox, then a message box with the message **Enter the subject** will be displayed.

Lines 18-21

If Txt_Send.Text = "" Then

MessageBox.Show("Enter the Text for the Message", "CADCIM")

Exit Sub

End If

In these lines, if the null string is assigned to the **Txt_Send** textbox, then the message **Enter the text for the message** will be displayed.

Line 23

msg.Priority = MessagePriority.Normal

In this line, the priority of the messages will be set. This indicates that the messages will be placed in the queue according to the priority.

Line 24

msg.Label = Txt_Label.Text

In this line, the text entered in the **Txt_Label** textbox will be assigned to the **Label** property of **msg**.

Line 25

msg.Body = Txt_Send.Text

In this line, the text entered in the **Txt_Send** textbox will be assigned to the **Body** property of **msg**.

Line 26

MsgQ.Send(msg)

In this line, the message will be sent within the local server.

Lines 28-30

For Each msg In MsgQ

CmbQueue.Items.Add(msg.Label)

Next

In these lines, each message entered by the user will be added to the **ComboBox** control.

Lines 31-34

Catch ex As Exception

MessageBox.Show("Error: " + ex.ToString())

End Try

End Sub

In these lines, **Catch** will catch the exception, if any. A message box with a specified text will be displayed. The **EndTry** statement indicates the end of the **Try** statement.

Line 37

Private Sub CmbQueue_SelectedIndexChanged(ByVal sender As System.Object, _

ByVal e As System.EventArgs) Handles CmbQueue.SelectedIndexChanged

In this line, the **SelectedIndexChanged** event of the **ComboBox** control is declared.

Line 38

Dim str() As String = {"System.String,mscorlib"}

In this line, the variable **str()** is declared as a string data type.

Line 39

MsgQ.Formatter = New XmlMessageFormatter(str)

XmlMessageFormatter is the default formatter that is used by an instance of **MessageQueue** to serialize the messages written for the queue. The message is then assigned to the formatter of **MessageQueue**.

Line 40

Dim Mymessage As Message = MsgQ.Receive()

In this line, **MsgQ.Receive()** will receive the message first and then assign it to the variable **Message**.

Step 5

Execute the application; the output form will be displayed. Enter message such as **Hello** in the **Subject** textbox and **How are you?** in the **Text Message** textbox. Choose the **Send Message** button; a message box with the message **Message Sent.** will be displayed, as shown in Figure 7-27. To view the message, select the subject from the **Select the Message** combo box; the message you have sent will be displayed in the textbox, as shown in Figure 7-28.

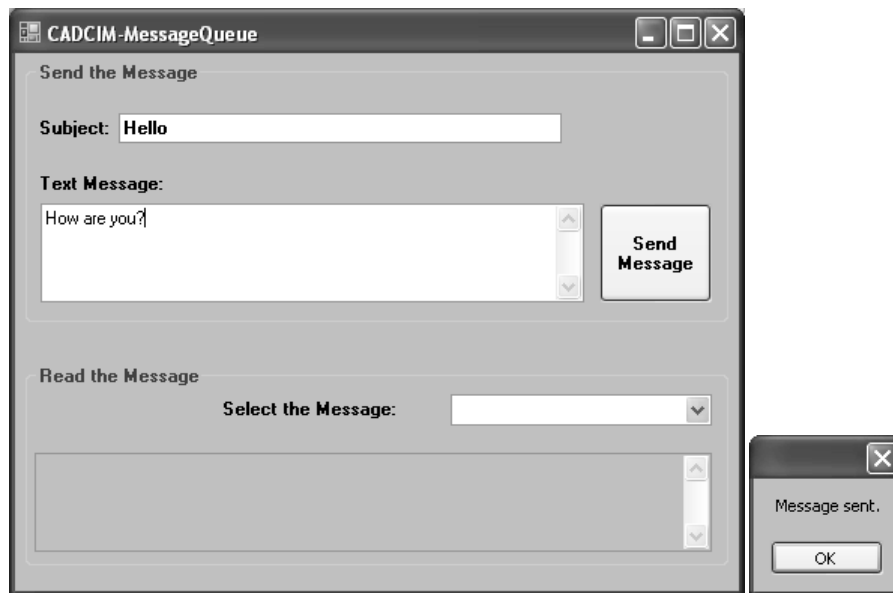


Figure 7-27 The output of Application 7 in case a message is sent



Figure 7-28 The output of Application 7 in case a message is retrieved

Self-Evaluation Test

Answer the following questions and then compare them to those given at the end of this chapter:

1. The _____ component is used to execute the time-consuming operations asynchronously in the background of an application.
2. The _____ component is used to validate the user's input on a control.
3. The _____ property determines whether the component is active or not.
4. The _____ component is used to store the collection of images.
5. The _____ component is used to access the processes on both the remote and local computers.

Review Questions

Answer the following questions:

1. The _____ is a standard method used by an application in your system to store a record of some software and hardware related events.
2. The _____ component is used to determine the conditions due to which an error has occurred.
3. The _____ property is used to import an image to the form.
4. With the help of the _____ component, an application can create and delete queues, explore the existing queues, send and receive messages to a queue on the **Message Queuing** servers, and so on.
5. You can use the _____ method to create a new event source that will be associated with an event log.

Exercise

Exercise 1

Create an application to display any process on your system using the **Process** component. The output of your application will be similar to Figure 7-29. The process should start when you choose the **Start** button and stop when you choose the **Stop** button.

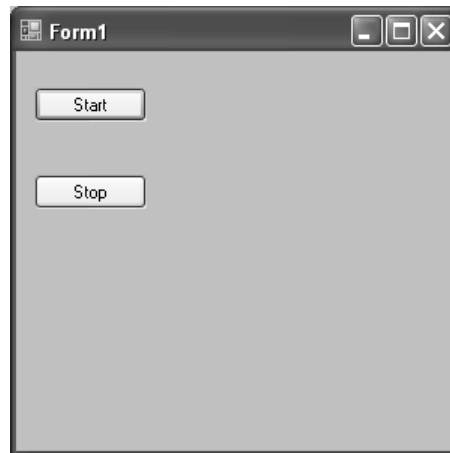


Figure 7-29 The output of Exercise 1

Answers to Self-Evaluation Test

1. BackgroundWorker, 2. ErrorProvider, 3. EnableRaisingEvents, 4. ImageList, 5. Process