**Table of Contents**

[Part IV – Programming with the .NET Framework 3](#_17dp8vu)

[System.Console 8](#_3rdcrjn)

[System.Windows.Forms 11](#_26in1rg)

[System.Drawing 25](#_lnxbz9)

[System.Data 34](#_35nkun2)

[System.Xml 48](#_1ksv4uv)

# **Part IV – Programming with the .NET Framework**

Having Visual Basic without the ".NET Framework" is about as useful as a pen without ink. You can write all you wish, but you’re not going to achieve anything useful.

What is the .NET framework? It is a programming platform developed by Microsoft. Visual Basic was specifically written as a language that will use the .NET Framework. The framework is made up of two different things:

1. A huge library of code that we can call from our Visual Basic programs. This saves us writing everything ourselves.
2. A "runtime" module, which runs our programs for us when we’re ready (this happens invisibly - you don't need to worry about it)

When you write a Visual Basic program (or a program in any of the other .NET languages), you typically call some code that lives in the library, as well as writing some of your own code.



There are so many classes in the .NET framework, and some of them are pretty complicated, so we certainly won't try to cover the whole thing here. Instead, this section's chapters go through some of the .NET framework classes that we think you'll find most useful. As you grow in experience, you'll find that there are certain classes that you use often and you'll get to know those ones well - like taking a favorite book out of the library - eventually you know just where to find it.

It's very important to realize that part III is not just for reading - these chapters include a lot of sample programs and you're supposed to **try them all out for yourself** and then try changing them to do **new things**.

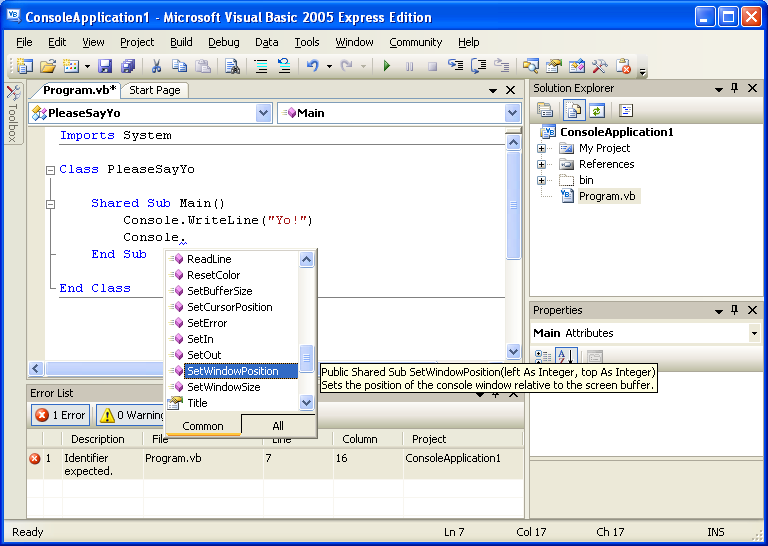
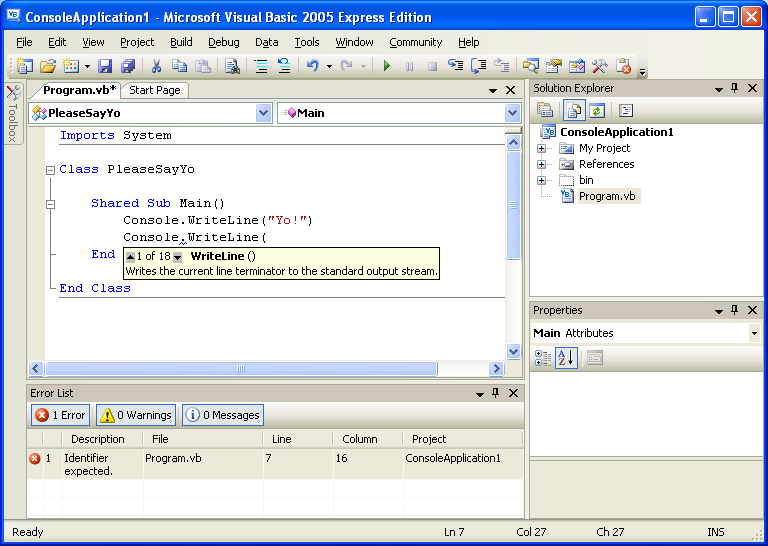
##### **How to Try Out the Examples**

In the following chapters, whenever you see a block with the words "code for program ..." you can run that code using Visual Basic Express. Although most of these programs are short, they are all real working programs. Here are the steps you will normally follow:

* Open Visual Basic Express. Select File -> Open Project and then browse to wherever you saved the example programs. Open any file with a "**.vbproj**" extension.  
  (or you can use Windows Explorer to find the sample you want and simply double-click the .vbproj file.)
* Press F5 to run the program (or click the green arrow)

##### **How to Change and Extend the Examples**

It's fine telling you "you can change the examples", but how do you know what to change them to? The examples show a few .NET Framework classes and methods being used - how do you know what other ones are available? There are a few ways to find out:

* Look through the .NET Framework SDK Class Library included in Visual Basic Express's help.
* While programming in Visual Basic Express, to see the available classes, methods, fields, etc for a class, press the "." key directly after typing a namespace or class name. E.g. Type "Console." and you'll see a list of methods and other members available in the Console class.  
    
    
    
  In the case of a method, if you type an open parenthesis at the end of the method name, you can also see the parameter types that the method expects. Often a method is written in a special way to allow different sets of parameters - then you can scroll through them by pressing the down and up arrow keys. The following example shows what happens when you type "Console.WriteLine(". Visual Basic Express shows that there are 18 different ways you can call WriteLine. In the picture below, we pressed the down arrow until we reached the 14th one.  
    
  



### **System.Console**

##### **Introduction**

The idea of a "Console" comes from the days when large mainframe computers were very popular. A company would have a giant computer hiding in some back room and the people outside would have a keyboard and a simple screen, called a console, which was hooked into the beast in the back room. These screens could not display true graphics – only text. The keyboard was the input device, sending information to the computer, and the console was the main output device, allowing to computer to send information back to the user.



The world moved on and these days most computers use displays that can show far more natural representations to people than lines of text – photographs for example.

There is often a need, however, to do things that don’t need to show anything fancy to the user. You may, for example, have a program which goes and fetches some data from an Internet server somewhere and puts it into a file on your pc. If all you need it to say is "busy fetching data" and then "done", why waste a whole lot of time and memory on a fancy user interface? It is for this reason that the .NET library offers us a class for easily writing console applications.

Don’t scoff at console applications, as if "they aren’t cool". In fact, you’ll find the really smart programmers get tired of wasting time with fancy interfaces and do most of their own work in console applications.

Of course, if you plan to write a program that someone else is going to use, you probably want to be a little kinder and give them something friendlier than a console interface.

Some Useful Methods

1. **Console.ReadLine** – reads a line of text from the keyboard (or some other input device)
2. **Console.Read** – reads a number from keyboard (or some other input device)
3. **Console.WriteLine** – writes text to the screen (or other output device), starting on a new line
4. **Console.Write** – writes some characters to the screen without starting a new line

##### **Example Program 1**

The following program simply writes the word "Yo!" to the screen and then waits for the ENTER key to be pressed.

| Code for program 1 |
| --- |
| Imports System  Class PleaseSayYo  Shared Sub Main()  ' Write a word to the screen  Console.Write("Yo!")  ' Wait until enter is pressed  Console.ReadLine()  End Sub  End Class |

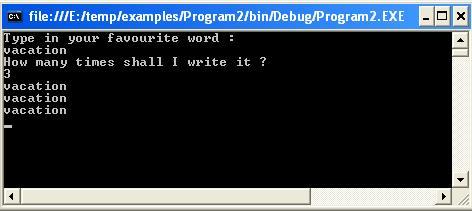
##### 

##### **Example Program 2**

##### The following program

* Asks the user to type in a word on the keyboard
* Asks the user how many times the word should be written out
* Writes out, on a new line each time, the word that was entered, as many times as was requested

| Code for program 2 |
| --- |
| Imports System  Class WriteAFewLines  Shared Sub Main()  Console.WriteLine("Type in your favourite word : ")  ' Store, in a string variable, the word that the user entered  Dim favouriteWord As String = Console.ReadLine()  Console.WriteLine("How many times shall I write it? ")  ' Store, in an integer variable, the number they type in  ' (And they'll get an error if they don't type a true number)  Dim numberOfTimes As Integer = Convert.ToInt32(Console.ReadLine())  ' Write the word out as many times as was requested  For i As Integer = 0 To numberOfTimes - 1  Console.WriteLine(favouriteWord)  Next  ' Wait until ENTER is pressed  Console.ReadLine()  End Sub  End Class |



### **System.Windows.Forms**

##### **Introduction**

If you want to write programs that look and feel like the ones you’re used to using in a Windows environment, you’ll most definitely want to use the classes in the System.Windows.Forms namespace. They allow you to work with buttons, list boxes, text boxes, menus, message boxes and a whole bunch of other "controls." Controls are things you place on a form – they either show things like text (a Label control) and pictures (a Picture Box control) or allow you to carry out actions such as selecting a value or clicking a button to move to another form. You’ll probably use the classes beneath System.Windows.Forms in most of your Visual Basic programs.

Obviously the idea of a "form" comes from the widely used paper form in the "real" world. A form is something which allows the placing of various things (text, pictures, entry boxes, etc.) in a well-organized layout. Generally, a person will *read* some information on the form and *fill in* some information in particular regions.



The idea on the computer is similar – a form allows the placing of text, pictures, entry boxes, buttons, etc, in a fashion which allows these to be precisely organized on the screen - very different to a console application, which can only handle lines of text following each other.

Microsoft has provided, in the .NET Framework class library, a huge number of "controls" for use on forms. Once you know how to place a control on a form, you can build up a snazzy application very quickly, simply by using these existing controls.

##### **Some useful classes in the System.Windows.Forms namespace**

The following are examples of classes with code for controls that you can place on your forms

* Label
* Button
* ListBox
* CheckBox
* RadioButton
* ListBox
* Menu
* TabControl
* Toolbar
* TreeView
* DataGrid
* PictureBox
* RichTextBox

##### **A note about working with the Windows Forms examples using Visual Basic Express**

To play with the following examples in Visual Basic Express, you can select File -> Open Project and open the book's sample .vbproj programs from wherever you chose to save them on your computer's disk.

If, however, you want to *type them yourself from scratch*, you need to be aware that when you create a new "Windows Application" project, Visual Basic Express puts down some **.vb** files for you (named **Form1.vb** and **Program.vb**), and inserts some Visual Basic code so that you're ready to go. It actually creates for you a basic but fully functional program. While you're working with the examples below, to keep things simple, you should probably

* delete Form1.vb and
* replace the code in Program.vb with the code from the example you're working with

This is not necessary if you rather open the examples using File -> Open Project.

##### **Example Program 3**

Here’s an absolutely simple Windows forms application. All it does is to start a new form and write some text in the titlebar of the window.

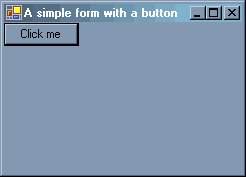
| Code for program 3 |
| --- |
| Imports System.Windows.Forms  Class SimpleWindowsForm  Inherits Form  ' The constructor method for our class  Public Sub New()  ' Set the window title  Me.Text = "A really Simple Form"  End Sub  Shared Sub Main()  ' Start a new instance of a forms application, using our class above  Application.Run(New SimpleWindowsForm())  End Sub  End Class |

##### 

##### **Example Program 4**

This example is simple too, but takes us to the next level – placing a button on the form

| Code for program 4 |
| --- |
| Imports System.Windows.Forms  Class SimpleWindowsFormWithButton  Inherits Form  Private button1 As Button  ' The constructor method for our class  Public Sub New()  ' Set the window title  Me.Text = "A simple form with a button"  ' Add a button to the form's collection of controls  ' Although the button says "click me", nothing happens when you do!  button1 = New Button()  button1.Text = "Click me"  Me.Controls.Add(button1)  End Sub  Shared Sub Main()  ' Start a new instance of a forms application, using our class above  Application.Run(New SimpleWindowsFormWithButton())  End Sub  End Class |

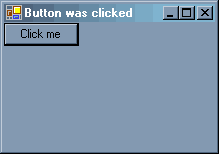


##### **Example Program 5**

Having a button on the form is okay, but in the example above, nothing happens when the user clicks the button. Boring.

We need to write a method that will do something when the button is clicked - let’s just make it change the Title Bar text in this case. We call such a method an event handler, since it will watch for an event (a click in this case) and will then decide how to handle it. We also need to hook the button click event up to the event handler.

| Code for program 5 |
| --- |
| Imports System  Imports System.Windows.Forms  Imports System.Drawing  Class FormWithWorkingButton  Inherits Form  ' The constructor method for our class  Public Sub New()  ' Set the window title  Me.Text = "Simple form with working button"  ' Add a button and hook it up to an event handler method  Dim button1 As Button = New Button()  button1.Text = "Click me"  AddHandler button1.Click, AddressOf Button1\_Click  Me.Controls.Add(button1)  End Sub  Shared Sub Main()  ' Start a new instance of a forms application, using our class above  Application.Run(New FormWithWorkingButton())  End Sub  ' An event handler that will be run when the button is clicked  Private Sub Button1\_Click(ByVal sender As Object, ByVal e As EventArgs)  ' Change the window title  Me.Text = "Button was clicked"  End Sub  End Class |



##### **Example Program 6**

Right, the program does all the basic stuff. Now let’s add a few other types of controls to the form, lay them out nicely and work with them a little. We'll use 4 control types: Button, ListBox, MessageBox and PictureBox.

Notice that, apart from System.Windows.Forms, we'll also use the System.Drawing namespace here. This is necessary because we're using a PictureBox - and working with images requires the Drawing classes.

| Code for program 6 |
| --- |
| Imports System.Windows.Forms  Imports System.Drawing  Class MyForm  Inherits Form  ' Declare a listbox outside the methods, since we'll want to talk to it  ' from more than one method  Private listBox1 As ListBox  ' The constructor method for our class  Public Sub New()  ' Create a picture box, put a picture in it and add it to this form  Dim pictureBox1 As PictureBox = New PictureBox()  pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage  Dim image1 As Bitmap = New Bitmap("../../images/seashore.jpg")  pictureBox1.ClientSize = New Size(Me.Width, 100)  pictureBox1.Image = DirectCast(image1, Image)  Me.Controls.Add(pictureBox1)  ' Create a button object,  ' set some of it's properties and add it to the form  Dim button1 As Button = New System.Windows.Forms.Button()  button1.Location = New System.Drawing.Point(10, 120)  button1.Text = "Click me"  AddHandler button1.Click, AddressOf button1\_Click  Me.Controls.Add(button1)  ' Create a listbox object,  ' set some of it's properties and add it to the form  listBox1 = New System.Windows.Forms.ListBox()  listBox1.Location = New System.Drawing.Point(10, 160)  listBox1.Items.Add("Sparkly")  listBox1.Items.Add("Dull")  listBox1.Items.Add("Vivacious")  listBox1.Items.Add("Plain")  listBox1.Items.Add("Outstanding")  listBox1.SelectedIndex = 3  Me.Controls.Add(listBox1)  End Sub  ' An event handler method that will run when the button is clicked  Private Sub button1\_Click(ByVal sender As Object, ByVal e As System.EventArgs)  ' Show a message indicating which item in the listbox is selected  MessageBox.Show(Me, "You have selected " \_  & listBox1.SelectedItem.ToString(), \_  "Notification", MessageBoxButtons.OK)  End Sub  Shared Sub Main()  ' Start a new instance of a forms application, using our class above  Application.Run(New MyForm())  End Sub  End Class |



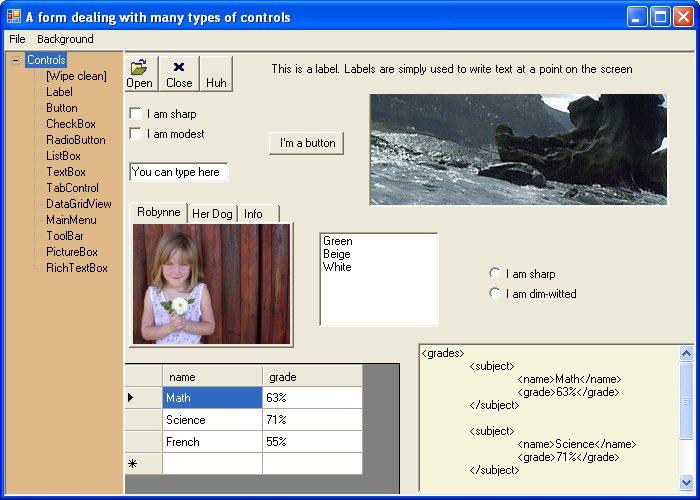
##### **Example Program 7**

Okay, now let’s get wild. To illustrate how to use some of the other controls, we’ll write one really large program including many useful controls. This will make the code **scarily long**, but it will be a useful program for you to refer back to when you need to use a particular control.

**You don't have to read the entire program in detail**, but when you're interested in using, for example, a CheckBox, come back to this program, find the parts that talk about CheckBox and study those parts.

Notice that in order to use the PictureBox and the DataGridView in interesting ways, we're also going to use the namespaces System.Drawing, System.Data and System.Xml.

| Code for program 7 |
| --- |
| Imports System  Imports System.Windows.Forms  Imports System.Drawing  Imports System.Data  Imports System.Xml  Class FormWithManyControls  Inherits Form  Private treeView1 As TreeView  Private panel1 As Panel  Private checkBox1 As CheckBox, checkBox2 As CheckBox  Private radiobutton1 As RadioButton, radioButton2 As RadioButton  Private listBox1 As ListBox  ' The constructor method for our class  Public Sub New()  ' Set window title and size  Me.Text = "A form dealing with many types of controls"  Me.Width = 700  Me.Height = 500  ' Add a tree view as a kind of menu  treeView1 = New TreeView()  treeView1.BackColor = Color.BurlyWood  treeView1.Dock = DockStyle.Left  AddHandler treeView1.AfterSelect, AddressOf treeView1\_AfterSelect  Dim tn As TreeNode = New TreeNode("Controls")  tn.Expand()  tn.Nodes.Add(New TreeNode("[Wipe clean]"))  tn.Nodes.Add(New TreeNode("Label"))  tn.Nodes.Add(New TreeNode("Button"))  tn.Nodes.Add(New TreeNode("CheckBox"))  tn.Nodes.Add(New TreeNode("RadioButton"))  tn.Nodes.Add(New TreeNode("ListBox"))  tn.Nodes.Add(New TreeNode("TextBox"))  tn.Nodes.Add(New TreeNode("TabControl"))  tn.Nodes.Add(New TreeNode("DataGridView"))  tn.Nodes.Add(New TreeNode("MainMenu"))  tn.Nodes.Add(New TreeNode("ToolBar"))  tn.Nodes.Add(New TreeNode("PictureBox"))  tn.Nodes.Add(New TreeNode("RichTextBox"))  treeView1.Nodes.Add(tn)  Me.Controls.Add(treeView1)  ' Add a panel to put all the other controls on  panel1 = New Panel()  panel1.Dock = DockStyle.Right  panel1.BorderStyle = BorderStyle.Fixed3D  panel1.Width = Me.Width - treeView1.Width  Me.Controls.Add(panel1)  End Sub  ' An eventhandler that runs when something in the treeview is selected  Private Sub treeView1\_AfterSelect(ByVal sender As Object, \_  ByVal e As System.Windows.Forms.TreeViewEventArgs)  ' Do something appropriate for whichever node was clicked  If e.Node.Text = "[Wipe clean]" Then  ' Wipe the panel clean of any controls  panel1.Controls.Clear()  ElseIf e.Node.Text = "Button" Then  ' Add a button to the panel  Dim button1 As Button = New Button()  button1.Text = "I'm a button"  button1.Location = New Point(150, 80)  AddHandler button1.Click, AddressOf button1\_Click  panel1.Controls.Add(button1)  ElseIf e.Node.Text = "Label" Then  ' Add a label to the panel  Dim label1 As Label = New Label()  label1.Text = \_  "This is a label. Labels are simply used " \_  & "to write text at a point on the screen"  label1.Location = New Point(150, 10)  label1.Width = 400  AddHandler label1.Click, AddressOf label1\_Click  panel1.Controls.Add(label1)  ElseIf e.Node.Text = "CheckBox" Then  ' Add some checkboxes to the panel  checkBox1 = New CheckBox()  checkBox1.Text = "I am sharp"  checkBox1.Location = New Point(10, 50)  AddHandler checkBox1.CheckedChanged, AddressOf CheckBox\_CheckedChanged  panel1.Controls.Add(checkBox1)  checkBox2 = New CheckBox()  checkBox2.Text = "I am modest"  checkBox2.Location = New Point(10, 70)  AddHandler checkBox2.CheckedChanged, AddressOf CheckBox\_CheckedChanged  panel1.Controls.Add(checkBox2)  ElseIf e.Node.Text = "RadioButton" Then  ' Add some radio buttons to the panel  radiobutton1 = New RadioButton()  radiobutton1.Text = "I am sharp"  radiobutton1.Location = New Point(370, 210)  AddHandler radiobutton1.CheckedChanged, \_  AddressOf RadioButton\_CheckedChanged  panel1.Controls.Add(radiobutton1)  radioButton2 = New RadioButton()  radioButton2.Text = "I am dim-witted"  radioButton2.Location = New Point(370, 230)  AddHandler radiobutton1.CheckedChanged, \_  AddressOf RadioButton\_CheckedChanged  panel1.Controls.Add(radioButton2)  ElseIf e.Node.Text = "ListBox" Then  ' Add a listbox to the panel  listBox1 = New ListBox()  listBox1.Items.Add("Green")  listBox1.Items.Add("Beige")  listBox1.Items.Add("White")  listBox1.Location = New Point(200, 180)  AddHandler listBox1.SelectedIndexChanged, \_  AddressOf listBox1\_SelectedIndexChanged  panel1.Controls.Add(listBox1)  ElseIf e.Node.Text = "TextBox" Then  ' Add a text box to the panel  Dim TextBox1 As TextBox = New TextBox()  TextBox1.Text = "You can type here"  TextBox1.Location = New Point(10, 110)  panel1.Controls.Add(TextBox1)  ElseIf e.Node.Text = "DataGridView" Then  ' Add a datagrid, populated with data from an xml file, to the panel  Dim dataSet1 As DataSet = New DataSet("A sample DataSet")  dataSet1.ReadXml("../../data/grades.xml")  Dim dataGridView1 As DataGridView = New DataGridView()  dataGridView1.Width = Convert.ToInt32((panel1.Width / 2) - 10)  dataGridView1.Height = 150  dataGridView1.Location = \_  New Point(2, panel1.Height - dataGridView1.Height - 5)  dataGridView1.DataSource = dataSet1  dataGridView1.DataMember = "subject"  panel1.Controls.Add(dataGridView1)  ElseIf e.Node.Text = "TabControl" Then  ' Add a tab control to the panel  ' and add some content to each tab page  Dim tabControl1 As TabControl = New TabControl()  tabControl1.Location = New Point(10, 150)  tabControl1.Size = New Size(165, 146)  Dim tabPage1 As TabPage = New TabPage("Robynne")  Dim pictureBox1 As PictureBox = New PictureBox()  pictureBox1.Image = New Bitmap("../../images/robynne.jpg")  pictureBox1.Size = New Size(160, 120)  tabPage1.Controls.Add(pictureBox1)  tabControl1.TabPages.Add(tabPage1)  Dim tabPage2 As TabPage = New TabPage("Her Dog")  Dim pictureBox2 As PictureBox = New PictureBox()  pictureBox2.Image = New Bitmap("../../images/chocolate.jpg")  pictureBox2.Size = New Size(160, 120)  tabPage2.Controls.Add(pictureBox2)  tabControl1.TabPages.Add(tabPage2)  Dim tabPage3 As TabPage = New TabPage("Info")  tabPage3.BackColor = Color.White  Dim label1 As Label = New Label()  label1.Text = "Robynne lives in Cape Town, South Africa." \_  & Environment.NewLine & Environment.NewLine \_  & "She has a dog named Chocolate, from the planet Woof," \_  & " rapidly growing into her oversized ears."  label1.Dock = DockStyle.Fill  tabPage3.Controls.Add(label1)  tabControl1.TabPages.Add(tabPage3)  panel1.Controls.Add(tabControl1)  ElseIf e.Node.Text = "PictureBox" Then  ' Add a picture to the panel  Dim pictureBox1 As PictureBox = New PictureBox()  pictureBox1.Image = New Bitmap("../../images/ocean.jpg")  pictureBox1.BorderStyle = BorderStyle.Fixed3D  pictureBox1.Location = New Point(250, 25)  pictureBox1.Size = New Size(300, 130)  panel1.Controls.Add(pictureBox1)  ElseIf e.Node.Text = "RichTextBox" Then  ' Add a box for typing rich text in  ' Load some data from an XML file into it  Dim richTextBox1 As RichTextBox = New RichTextBox()  richTextBox1.LoadFile("../../data/grades.xml", \_  RichTextBoxStreamType.PlainText)  richTextBox1.WordWrap = False  richTextBox1.BorderStyle = BorderStyle.Fixed3D  richTextBox1.BackColor = Color.Beige  richTextBox1.Size = \_  New Size(Convert.ToInt32((panel1.Width / 2) - 10), 150)  richTextBox1.Location = \_  New Point(Convert.ToInt32((panel1.Width / 2) + 10), \_  panel1.Height - richTextBox1.Height - 5)  panel1.Controls.Add(richTextBox1)  ElseIf e.Node.Text = "MainMenu" Then  ' Add a classic "menu" (appears at the top of the window)  Dim mainMenu1 As MainMenu = New MainMenu()  Dim menuItem1 As MenuItem = New MenuItem("File")  menuItem1.MenuItems.Add("Exit", \_  New EventHandler(AddressOf mainMenu1\_Exit\_Select))  mainMenu1.MenuItems.Add(menuItem1)  Dim menuItem2 As MenuItem = New MenuItem("Background")  menuItem2.MenuItems.Add("Choose", \_  New EventHandler(AddressOf mainMenu1\_ColorOwn\_Select))  menuItem2.MenuItems.Add("White", \_  New EventHandler(AddressOf mainMenu1\_ColorWhite\_Select))  mainMenu1.MenuItems.Add(menuItem2)  Me.Menu = mainMenu1  MessageBox.Show("A main menu has been added at the top left of the " \_  & " window. Try it out after clicking OK.")  ElseIf e.Node.Text = "ToolBar" Then  ' Add a shortcuts toolbar to the panel  Dim toolBar1 As ToolBar = New ToolBar()  Dim imageList1 As ImageList = New ImageList()  imageList1.Images.Add(New Bitmap("../../images/open.gif"))  imageList1.Images.Add(New Bitmap("../../images/close.gif"))  imageList1.Images.Add(New Bitmap("../../images/undo.gif"))  toolBar1.ImageList = imageList1  Dim toolBarbutton1 As ToolBarButton = New ToolBarButton("Open")  toolBarbutton1.ImageIndex = 0  toolBar1.Buttons.Add(toolBarbutton1)  Dim toolBarbutton2 As ToolBarButton = New ToolBarButton("Close")  toolBarbutton2.ImageIndex = 1  toolBar1.Buttons.Add(toolBarbutton2)  Dim toolBarButton3 As ToolBarButton = New ToolBarButton("Huh")  toolBarButton3.ImageIndex = 3  toolBar1.Buttons.Add(toolBarButton3)  AddHandler toolBar1.ButtonClick, AddressOf toolBar1\_Click  panel1.Controls.Add(toolBar1)  End If  End Sub  ' All other event handlers - for the controls added above  ' An eventhandler that will be run if the label is clicked  Private Sub label1\_Click(ByVal sender As Object, ByVal e As System.EventArgs)  MessageBox.Show("Yes, labels can be clicked, " \_  & "although it's not normal to do so.")  End Sub  ' An eventhandler that will be run if the button is clicked  Private Sub button1\_Click(ByVal sender As Object, ByVal e As System.EventArgs)  MessageBox.Show("Yup, you were supposed to click me")  End Sub  ' An eventhandler that will be run if a checkbox is checked/unchecked  Private Sub CheckBox\_CheckedChanged(ByVal sender As Object, \_  ByVal e As System.EventArgs)  If checkBox1.Checked AndAlso checkBox2.Checked Then  MessageBox.Show("Good for you")  ElseIf checkBox1.Checked Then  MessageBox.Show("It's not good to be sharp without being modest")  ElseIf checkBox2.Checked Then  MessageBox.Show("Modesty is good. Pity you're not sharp too.")  Else  MessageBox.Show("Oh dear, neither sharp nor modest eh?")  End If  End Sub  ' An eventhandler that will be run if a radiobutton is clicked  Private Sub RadioButton\_CheckedChanged(ByVal sender As Object, \_  ByVal e As System.EventArgs)  If radiobutton1.Checked Then  MessageBox.Show("Glad to hear it")  ElseIf radioButton2.Checked Then  MessageBox.Show("What a shame")  End If  End Sub  ' An eventhandler that will be run if an item in the listbox is selected  Private Sub listBox1\_SelectedIndexChanged(ByVal sender As Object, \_  ByVal e As System.EventArgs)  If listBox1.SelectedItem.ToString() = "Green" Then  treeView1.BackColor = Color.LightSeaGreen  ElseIf listBox1.SelectedItem.ToString() = "Beige" Then  treeView1.BackColor = Color.Beige  ElseIf listBox1.SelectedItem.ToString() = "White" Then  treeView1.BackColor = Color.White  End If  End Sub  ' An eventhandler that will be run if "white" is selected at the menu  Private Sub mainMenu1\_ColorWhite\_Select(ByVal sender As Object, \_  ByVal e As System.EventArgs)  treeView1.BackColor = Color.White  End Sub  ' An eventhandler that will be run if a color is selected at the menu  Private Sub mainMenu1\_ColorOwn\_Select(ByVal sender As Object, \_  ByVal e As System.EventArgs)  Dim colorDialog1 As ColorDialog = New ColorDialog()  colorDialog1.Color = treeView1.BackColor  colorDialog1.ShowDialog()  treeView1.BackColor = colorDialog1.Color  End Sub  ' An eventhandler that will be run if the user chooses "exit" at the menu  Private Sub mainMenu1\_Exit\_Select(ByVal sender As Object, \_  ByVal e As System.EventArgs)  If MessageBox.Show("Are you sure you want to exit?", \_  "Exit confirmation", MessageBoxButtons.YesNo) \_  = Windows.Forms.DialogResult.Yes Then  Me.Dispose()  End If  End Sub  ' An eventhandler that will be run if a toolbar shortcut is selected  Private Sub toolBar1\_Click(ByVal sender As Object, \_  ByVal e As System.Windows.Forms.ToolBarButtonClickEventArgs)  If e.Button.Text = "Open" Then  MessageBox.Show("This could have opened a file, for example")  ElseIf e.Button.Text = "Close" Then  MessageBox.Show("This could have closed a file, for example")  ElseIf e.Button.Text = "Huh" Then  MessageBox.Show("Huh?")  End If  End Sub  Shared Sub Main()  ' Start a new instance of a forms application, using our class above  Application.Run(New FormWithManyControls())  End Sub  End Class |



### **System.Drawing**

**Introduction**

The classes grouped together in the "Drawing" namespace allow us to work with different types of pictures. On computers, we usually deal with two general types of pictures:

| **Bitmap or Raster graphics** | **Vector graphics** |
| --- | --- |
|  |  |
| Bitmap graphics are pictures made up of many dots. For example, photographs and icons can be represented well as bitmaps. | Vector graphics are pictures made up of specific shapes like lines, circles, rectangles, etc. A house plan, for example, can be nicely represented using vector graphics. |

#### **Bitmap Graphics**

First, we'll show some examples of how to work with bitmap graphics. It is often useful to work with pictures such as photographs on the computer, and the .NET Framework class library includes quite a lot of useful code for doing so.

**Example Program 8**

This program simply fetches a bitmap image (a JPEG file in this case) from the disk and displays it on a form.

To display an image on a form, it is helpful to use some control that is capable of displaying images. The PictureBox control is perfect for this purpose.

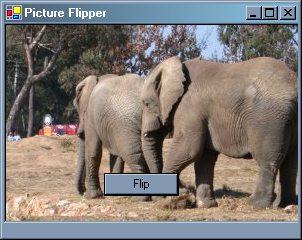
| Code for program 8 |
| --- |
| Imports System  Imports System.Windows.Forms  Imports System.Drawing  Class PictureDisplayer  Inherits Form  Private image1 As Bitmap  Private pictureBox1 As PictureBox  ' The constructor method for our class  Public Sub New()  ' Set the window title and size  Me.Text = "Picture Displayer"  Me.Size = New Size(302, 240)  ' Prepare a box to hold a picture  pictureBox1 = New PictureBox()  pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage  pictureBox1.BorderStyle = BorderStyle.Fixed3D  pictureBox1.ClientSize = New Size(300, 196)  ' Add a picture to the box  image1 = New Bitmap("../../images/lama.jpg")  pictureBox1.Image = DirectCast(image1, Image)  ' Add the box (with it's picture) to the form  Me.Controls.Add(pictureBox1)  End Sub  Shared Sub Main()  ' Start a new instance of a forms application, using our class above  Application.Run(New PictureDisplayer())  End Sub  End Class |

****

**Example Program 9**

This next program loads a photograph from disk and then also allows the user to flip it horizontally by clicking the "flip" button.

| Code for program 9 |
| --- |
| Imports System  Imports System.Windows.Forms  Imports System.Drawing  Class PictureFlipper  Inherits Form  Private button1 As Button  Private image1 As Bitmap  Private pictureBox1 As PictureBox  ' The constructor method for our class  Public Sub New()  ' Set the window title and size  Me.Text = "Picture Flipper"  Me.Size = New Size(302, 240)  ' Add a button to the form  button1 = New Button()  button1.Text = "Flip"  button1.Location = New Point(100, 150)  AddHandler button1.Click, AddressOf button1\_Click  Me.Controls.Add(button1)  ' Add a picture box to the form  pictureBox1 = New PictureBox()  pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage  pictureBox1.BorderStyle = BorderStyle.Fixed3D  pictureBox1.ClientSize = New Size(300, 196)  ' Add an image to the picture box  image1 = New Bitmap("../../images/elephants.jpg")  pictureBox1.Image = DirectCast(image1, Image)  ' Add the picture box (with its image) to the form  Me.Controls.Add(pictureBox1)  End Sub  Shared Sub Main()  ' Start a new instance of a forms application, using our class above  Application.Run(New PictureFlipper())  End Sub  ' An eventhandler that will run when the button is clicked  Private Sub button1\_Click(ByVal sender As Object, ByVal e As EventArgs)  ' Flip the image along the X axis (horizontally)  image1.RotateFlip(RotateFlipType.RotateNoneFlipX)  ' Re-insert the image into the picture box  pictureBox1.Image = DirectCast(image1, Image)  ' Update the window title  Me.Text = "Picture was flipped"  End Sub  End Class |

****

#### **Vector Graphics**

Now let's move on to examples dealing with vector graphics - pictures that are made up of specific shapes.

In all of these examples, we'll create a button and an event handler method to catch the button's click event. Only once the button is clicked will we run the code that works with the graphics.

Here are some important concepts to understand. They're very logical - but if you didn't know these you might feel a little confused:

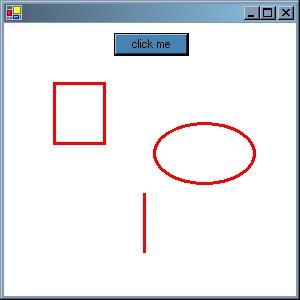
1. In the real world, to draw a line, circle, rectangle, etc, you first need to select a pen of the correct color and thickness.  
     
   Similarly, to draw a plain shape on the computer, you must first create **a Pen object**. For example, this code creates a Pen object that will draw in green with a thickness of 3 pixels :  
     
   Dim myGreenPen As Pen = New Pen(Color.Green,3)
2. To create a colored-in shape, you could use something like a paintbrush.  
     
   On the computer, **colored-in shapes** can only be created if you have created **a Brush object** with some chosen color. There are different types of brushes available; the following piece of code will create a blue SolidBrush object :

Dim *myBlueBrush* As SolidBrush = New SolidBrush(Color.Blue)

**Example Program 10**

In this program, the DrawSomeShapes method creates a line, a rectange and an ellipse (a squashed circle).

| Code for program 10 |
| --- |
| Imports System  Imports System.Windows.Forms  Imports System.Drawing  Class SimpleShapeMaker  Inherits Form  ' The constructor method for our class  Public Sub New()  ' Set the form background to white  Me.BackColor = Color.White  ' Add a button, hooked up to an event handler, to the form  Dim button1 As Button = New Button()  button1.Text = "click me"  button1.Location = New Point(110, 10)  button1.BackColor = Color.SteelBlue  AddHandler button1.Click, AddressOf button1\_Click  Me.Controls.Add(button1)  End Sub  ' An eventhandler that will run when the button is clicked  Private Sub button1\_Click(ByVal o As Object, ByVal e As System.EventArgs)  ' Execute a specific method we wrote  DrawSomeShapes()  End Sub  ' A method that draws a few shapes on the form surface  Private Sub DrawSomeShapes()  ' Prepare a drawing surface for this form  Dim g As Graphics = Me.CreateGraphics()  ' Prepare a pen that draws red and with a width of 3  Dim redPen As Pen = New Pen(Color.Red, 3)  ' Use the pen to draw a straight line, a rectangle and an oval  g.DrawLine(redPen, 140, 170, 140, 230)  g.DrawRectangle(redPen, 50, 60, 50, 60)  g.DrawEllipse(redPen, 150, 100, 100, 60)  ' Clean up  g.Dispose()  End Sub  Shared Sub Main()  ' Start a new instance of a forms application, using our class above  Application.Run(New SimpleShapeMaker())  End Sub  End Class |



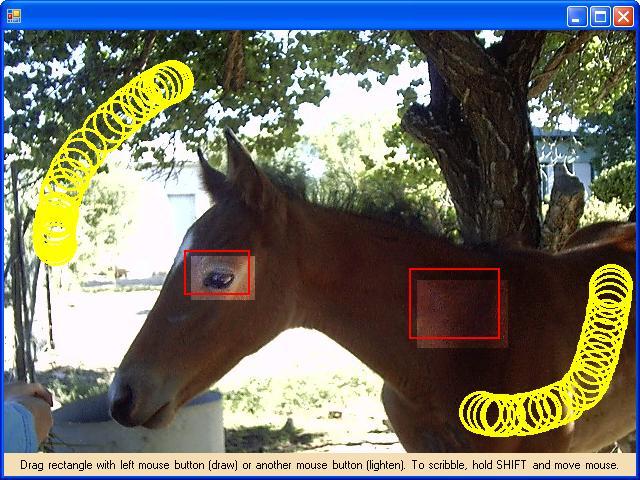
**Example Program 11**

How about we have a bit of fun with the mouse now? Doing interesting graphical things is usually easier with a mouse than with a keyboard. In this next example, we work with both bitmap and vector graphics but also take the opportunity to use a few mouse events.

We do a few interesting new things below - particularly with bitmaps. Although we don’t want to write an essay about it, here’s a *little bit* of background to help you understand the concepts behind the code:

* Computer programs make graphics appear on the screen by changing the color and brightness of tiny dots called *pixels*.
* Each pixel is made up of the three primary colors: red, green and blue (often shortened to RGB in programming languages). You change the color/brightness of the pixel by varying the strengths of the R, G and B, typically between the values of 0 and 255. For example:
  + if red=255 and green=0 and blue=0, the pixel will appear bright red.
  + if red=255 and green=255 and blue=0, the pixel will appear yellow.
* The mouse’s position can be detected by the computer and is specified in terms of X and Y co-ordinates (horizontal and vertical co-ordinates). The top left of the screen, for example, is specified by X=0 and Y=0.

| Code for program 11 |
| --- |
| Imports System  Imports System.Windows.Forms  Imports System.Drawing  Class FunWithTheMouse  Inherits Form  ' Declare some objects we'll want to talk to from a few methods  Private pictureBox1 As PictureBox  Private label1 As Label  Private spotClicked As Point  ' The constructor method for our class  Public Sub New()  ' Set the window size  Me.Size = New Size(640, 480)  ' Load a photo into a picture box and add it to the form  pictureBox1 = New PictureBox()  pictureBox1.Image = DirectCast(New Bitmap("../../images/foal.jpg"), Image)  pictureBox1.SizeMode = PictureBoxSizeMode.Normal  pictureBox1.Dock = DockStyle.Fill  Me.Controls.Add(pictureBox1)  ' Add a label to the bottom, carrying some instructions  label1 = New Label()  label1.BackColor = Color.Wheat  label1.Dock = DockStyle.Bottom  label1.Text = "Drag rectangle with left mouse button (draw) or another " \_  & "mouse button (lighten). To scribble, hold SHIFT and move mouse."  label1.TextAlign = ContentAlignment.MiddleCenter  Me.Controls.Add(label1)  ' Hook up the picturebox to event handlers for doing mousy things  AddHandler Me.pictureBox1.MouseDown, AddressOf MouseButtonIsDown  AddHandler Me.pictureBox1.MouseUp, AddressOf MouseButtonIsUp  AddHandler Me.pictureBox1.MouseMove, AddressOf TheMouseMoved  End Sub  ' An event handler that will run when the mouse is MOVED  Public Sub TheMouseMoved(ByVal sender As Object, ByVal e As MouseEventArgs)  ' If the user is holding down a SHIFT key on the keyboard  If (Control.ModifierKeys And Keys.Shift) = Keys.Shift Then  ' Prepare a drawing surface on the picture  Dim g As System.Drawing.Graphics = Me.pictureBox1.CreateGraphics()  ' Use a yellow pen  Dim yellowPen As System.Drawing.Pen = \_  New System.Drawing.Pen(Color.Yellow, 3)  ' Draw a circle (an ellipse with equal width & height)  ' at the current X and Y co-ordinates of the mouse  g.DrawEllipse(yellowPen, e.X, e.Y, 40, 40)  ' Clean up  g.Dispose()  End If  End Sub  ' An event handler that will run when a mouse button is pressed DOWN  Public Sub MouseButtonIsDown(ByVal sender As Object, ByVal e As MouseEventArgs)  ' Just remember WHERE the mouse was clicked. When the  ' button is released, we'll want to retrieve this location  spotClicked.X = e.X ' horizontal co-ordinate  spotClicked.Y = e.Y ' vertical co-ordinate  End Sub  ' An event handler that will run when a mouse button is released  Public Sub MouseButtonIsUp(ByVal sender As Object, ByVal e As MouseEventArgs)  ' Ah, the user let go the button they were holding down!  ' Define a rectangle (not yet a visual one) that describes  ' the area of the picture the user wants to do something with  Dim r As Rectangle = New Rectangle()  ' The left, top corner is where they pressed the mouse down,  ' which we "remembered" in the method above  r.X = spotClicked.X  r.Y = spotClicked.Y  ' The width and height of the rectangle can be worked out  ' by subtracting the original point clicked from where  ' the mouse is now (when the user is letting go)  r.Width = e.X - spotClicked.X  r.Height = e.Y - spotClicked.Y  If e.Button = Windows.Forms.MouseButtons.Left Then  ' If left button was pressed/released, just draw  ' a visual rectangle so they can see it  ' Prepare a drawing surface on the picture box  Dim g As Graphics = Me.pictureBox1.CreateGraphics()  ' Draw a rectangle in red  Dim redPen As Pen = New Pen(Color.Red, 2)  g.DrawRectangle(redPen, r)  Else  ' If it was another button, call our more complicated  ' method that knows how to "lighten" an area of the photo  ChangeLightness(r)  End If  End Sub  ' A method that increases the lightness of the image portion  ' selected, by increasing the lightness of each pixel in that area  Public Sub ChangeLightness(ByVal rect As Rectangle)  Dim newRed As Integer, newGreen As Integer, newBlue As Integer  Dim pixel As Color  ' Get a copy of the picture that's in the picture box  Dim picture As System.Drawing.Bitmap = New Bitmap(Me.pictureBox1.Image)  ' Since brightening an area can take a long time,  ' warn the user about this if they select a big area  If (rect.Width > 150) OrElse (rect.Height > 150) Then  Dim result As DialogResult = MessageBox.Show( \_  "The area you selected is large and may take " \_  & " a long time to lighten", \_  "Warning", \_  MessageBoxButtons.OKCancel)  ' If they click "cancel", jump out of this method  ' and return to whatever called it  If result = Windows.Forms.DialogResult.Cancel Then  Return  End If  End If  ' Step through all pixels in the given block, doubling the  ' brightness of each pixel's RGB (red, green and blue) colors  ' From left to right, across the width of the area selected ...  For x As Integer = rect.X To rect.X + rect.Width - 1  ' And from top to bottom, down the height of the area selected ...  For y As Integer = rect.Y To (rect.Y + rect.Height) - 1  ' Read the pixel we're at  pixel = picture.GetPixel(x, y)  ' Prepare a brighter version of the three primary colors in that pixel  newRed = Convert.ToInt32(Math.Round(pixel.R \* 2, 0))  If newRed > 255 Then  newRed = 255  End If  newGreen = Convert.ToInt32(Math.Round(pixel.G \* 2, 0))  If newGreen > 255 Then  newGreen = 255  End If  newBlue = Convert.ToInt32(Math.Round(pixel.B \* 2, 0))  If newBlue > 255 Then  newBlue = 255  End If  ' Apply the new colors to the pixel  picture.SetPixel(x, y, \_  Color.FromArgb(Convert.ToByte(newRed), \_  Convert.ToByte(newGreen), \_  Convert.ToByte(newBlue)))  Next  Next  ' Put the modified copy of the picture into the pictureBox  ' so the user can see the changes  Me.pictureBox1.Image = picture  End Sub  Shared Sub Main()  ' Create a new instance of our form class  Application.Run(New FunWithTheMouse())  End Sub  End Class |



### **System.Data**

##### **Introduction**

Most applications out there need to work with **databases**. Ask any big company's programmers and you'll hear them speak about how important databases are in the computing world. A programmer who can work with databases will be in a position to create a great many really **useful** applications.

You might have a database such as Microsoft Access on your computer. Alternatively, you could install Microsoft SQL Server Express Edition, which is a really nice way to learn about the SQL Server database, used in many of the largest companies around the globe. SQL Server Express is available as part of the Visual Basic Express installation, so you may already have it installed.

The System.Data classes in the .NET Framework allow you to work with databases. A database is quite different to things like pictures and word processor documents, which are often called *unstructured*. A database is more **structured**. It most often contains many **rows** of the same type of data, grouped into blocks called **tables**. The table contains one or more **columns** and each column holds a particular piece of information for that row.

| Rows are sometimes called **records** and columns are sometimes called **fields**. |
| --- |

Here is a representation of a database table that holds information about planets. The columns in this case are PlanetName, DistanceFromSun and Inhabitants.

| **PLANET** | | |
| --- | --- | --- |
| PlanetName | DistanceFromSun | Inhabitants |
| Mercury | 57909 | Mercurians |
| Venus | 108200 | Venusians |
| Earth | 149600 | Earthlings |
| Mars | 227940 | Martians |
| Jupiter | 778400 | Jupiterians |
| Znock | 7208100 | Znockers |
| Saturn | 1423600 | Saturnians |
| Uranus | 2867000 | Uranians |
| Neptune | 4488400 | Neptunians |
| Pluto | 5909600 | Plutonians |

You can see, for example, that the planet Venus is 108 200 thousand kilometers from the sun and that the creatures living there are called Venusians.

Here's another table, this time showing how many creatures were found living on the planets each year.

*This is top secret information never revealed before. It was retrieved from an alien craft that crashed in a remote part of the Gobi desert. You should feel privileged to have a copy. Apparently, they too use SQL Server Express databases, which made it easier for us to bundle a copy with the code samples.*

| **POPULATION** | | |
| --- | --- | --- |
| PlanetName | Year | Population |
| Mercury | 2000 | 40000 |
| Venus | 2000 | 25 |
| Earth | 2000 | 6000000000 |
| Mars | 2000 | 325000 |
| Jupiter | 2000 | 8426300200 |
| Znock | 2000 | 550000 |
| Saturn | 2000 | 1000000 |
| Uranus | 2000 | 753425370 |
| Neptune | 2000 | <NULL> |
| Pluto | 2000 | <NULL> |
| Mercury | 2001 | 35000 |
| Venus | 2001 | 3 |
| Earth | 2001 | 6500000000 |
| Mars | 2001 | 326000 |
| Jupiter | 2001 | 8426300202 |
| Znock | 2001 | 8700 |
| Saturn | 2001 | 75000 |
| Uranus | 2001 | 844360002 |
| Neptune | 2001 | <NULL> |
| Pluto | 2001 | <NULL> |

Looking at all the rows that refer to Venus, you'll notice there are two. You can see that in the year 2000 there were 25 Venusians on Venus, but in 2001 there were only 3 of them left. I guess the volcanoes wiped them out.

| Don't confuse database tables with spreadsheets. While it's true that spreadsheets can show data in a way that looks like the tables above, the way they work with the data is quite different. |
| --- |

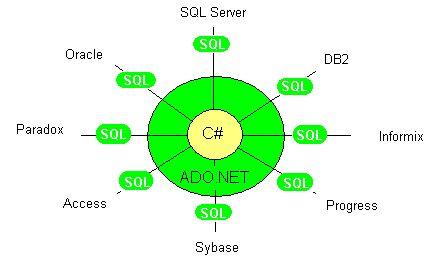
##### **The SQL Language**

There are many different databases around the world: Microsoft Access, Oracle, DB2, Microsoft SQL Server, Informix, mySQL and so the list goes on. So how do you talk to a database from Visual Basic? Will they all understand what we're asking for?

The simple answer is that you use a language like Visual Basic to wrap up and send some "database language" to the database, and it decides how to fetch and send back the columns and rows that you asked for.

(To tell the truth, there is another layer in-between called ADO.NET, but we won't talk too much about that here)

Many years ago, because of all the different databases, some people got together and agreed on "one database language" that could speak to most of the databases around. That language is called Structured Query Language (SQL for short). Don’t confuse the SQL *language* with Microsoft’s *product* named SQL Server – most databases support the SQL *language*.



Before we talk about how to work with databases in Visual Basic, let's get familiar with the basics of the SQL language. Here follow some examples of statements written in SQL and what happens when you run them.

The three main types of actions are SELECT to view some data, INSERT to insert new data, and UPDATE to change existing data - we'll give examples of each of these.

Usually the way you write Select statements is the following:

SELECT <the Columns you want to see>

FROM <the appropriate Database Tables>

WHERE <some condition is true>

**SELECT** \*

**FROM** PLANET

Brings back *all rows* and *all columns* from the table called PLANET.  
(The star \* means **all** columns)

| PlanetName | DistanceFromSun | Inhabitants |
| --- | --- | --- |
| Mercury | 57909 | Mercurians |
| Venus | 108200 | Venusians |
| Earth | 149600 | Earthlings |
| Mars | 227940 | Martians |
| Jupiter | 778400 | Jupiterians |
| Znock | 7208100 | Znockers |
| Saturn | 1423600 | Saturnians |
| Uranus | 2867000 | Uranians |
| Neptune | 4488400 | Neptunians |
| Pluto | 5909600 | Plutonians |

**SELECT** PlanetName, Inhabitants

**FROM** PLANET

Brings back just the "PlanetName" and "Inhabitants" columns for all rows in the PLANET table.

| PlanetName | Inhabitants |
| --- | --- |
| Mercury | Mercurians |
| Venus | Venusians |
| Earth | Earthlings |
| Mars | Martians |
| Jupiter | Jupiterians |
| Znock | Znockers |
| Saturn | Saturnians |
| Uranus | Uranians |
| Neptune | Neptunians |
| Pluto | Plutonians |

**SELECT** PlanetName, Inhabitants

**FROM** PLANET

**WHERE** PlanetName='Venus'

Brings back just the "PlanetName" and "Inhabitants" columns for only those rows in the PLANET table which have a PlanetName of "Venus".

| PlanetName | Inhabitants |
| --- | --- |
| Venus | Venusians |

**SELECT** PlanetName

**FROM** POPULATION

**WHERE** Population<100000

Brings back the PlanetName and Population, from the POPULATION table, wherever the population column has a value less than 100000.

| PlanetName | Population |
| --- | --- |
| Mercury | 40000 |
| Venus | 25 |
| Neptune | <NULL> |
| Pluto | <NULL> |
| Mercury | 35000 |
| Venus | 3 |
| Saturn | 75000 |
| Neptune | <NULL> |
| Pluto | <NULL> |

Usually the way you write Insert statements is the following:

INSERT INTO <the Database Table you want to add rows to>

(<the Columns you want to add values into>)

VALUES (<the values you want to add into the columns listed above>)

**INSERT** INTO PLANET

(PlanetName, DistanceFromSun, Inhabitants)

**VALUES**

('Fluff', 23500000, 'Fluffies')

Adds a new row to the PLANET table. This is actually a "silent" action - it doesn't bring back any rows to your Visual Basic program - but we show the table here so you get a picture of what's happened.

| **PLANET** | | |
| --- | --- | --- |
| PlanetName | DistanceFromSun | Inhabitants |
| Mercury | 57909 | Mercurians |
| Venus | 108200 | Venusians |
| Earth | 149600 | Earthlings |
| Mars | 227940 | Martians |
| Jupiter | 778400 | Jupiterians |
| Znock | 7208100 | Znockers |
| Saturn | 1423600 | Saturnians |
| Uranus | 2867000 | Uranians |
| Neptune | 4488400 | Neptunians |
| Pluto | 5909600 | Plutonians |
| Fluff | 23500000 | Fluffies |

Usually the way you write Update statements is the following:

UPDATE <the Database Table you want to change>

SET <Columns you want to change> = <new values>

WHERE <all the rows you want to change meet some condition>

**UPDATE** PLANET

**SET** PlanetName='Stuff', Inhabitants='Stuffies'

**WHERE** PlanetName='Fluff'

Changes some of the values in the row which has a PlanetName "Fluff". We show the resulting table here, but in reality this is a "silent" action and will not bring back any rows to your Visual Basic program.

| **PLANET** | | |
| --- | --- | --- |
| PlanetName | DistanceFromSun | Inhabitants |
| Mercury | 57909 | Mercurians |
| Venus | 108200 | Venusians |
| Earth | 149600 | Earthlings |
| Mars | 227940 | Martians |
| Jupiter | 778400 | Jupiterians |
| Znock | 7208100 | Znockers |
| Saturn | 1423600 | Saturnians |
| Uranus | 2867000 | Uranians |
| Neptune | 4488400 | Neptunians |
| Pluto | 5909600 | Plutonians |
| Stuff | 23500000 | Stuffies |

##### **Relationships and Joining**

If you think about it, you will notice there is a relationship between the two tables PLANET and POPULATION above. They both have a column called "PlanetName." We say that the two tables are **related** on the column "PlanetName" - and that allows us to collect all the information for a particular planet

We could take, for example, all the rows that have to do with Venus, from both tables ...

| | **PLANET** | | | | --- | --- | --- | | PlanetName | DistanceFromSun | Inhabitants | | Venus | 108200 | Venusians | | | **POPULATION** | | | | --- | --- | --- | | PlanetName | Year | Population | | Venus | 2000 | 25 | | Venus | 2001 | 3 | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |

and join them together into what appears to be one big table ...

**SELECT** \*

**FROM** PLANET INNER JOIN POPULATION ON PLANET.PlanetName=POPULATION.planetName

**WHERE** PlanetName='Venus'

| **PLANETS\_AND\_POPULATION** | | | | | |
| --- | --- | --- | --- | --- | --- |
| PlanetName | DistanceFromSun | Inhabitants | PlanetName | Year | Population |
| Venus | 108200 | Venusians | Venus | 2000 | 25 |
| Venus | 108200 | Venusians | Venus | 2001 | 3 |

##### **Which Database are you Using?**

There are reasons why programmers may want to do special things for special databases or situations. In the .NET environment, for example, there are several different ways to work with data. If you know you're using a Microsoft SQL Server database, for example, you can use special objects to send your SQL queries and because of that it will work really fast. But if you're talking to Microsoft Access, you can't use that special object.

The code differs slightly then, depending on whether you’re using SQL Server or not, and *we’re not sure whether you are*. So here’s what we’ve done:

* The three database example programs in this section (12a, 13a and 14a) are written assuming you *have* got SQL Server Express installed (or one of the other SQL Server versions).
* But in case you haven’t, we’ve also included, with the disk samples, a version of each that uses Microsoft Access. These are programs 12b, 13b and 14b, and they will run without needing any database setup at all.

We encourage you to install SQL Server Express at some stage though – it’s a much better database to program against. Additionally, SQL Server skills are more valued in the business world – so the sooner you get to know SQL Server the better. You can download it free from <http://msdn.microsoft.com/vstudio/express/sql/download/>.

| If you have Microsoft SQL Server Express installed and working, use examples 12a, 13a and 14a. If you don’t, or if you have trouble getting them working, you can fall back to examples 12b, 13b and 14b instead, which do the same thing without needing a database installed. |
| --- |

##### **Talking to a Database from Visual Basic**

In the following Visual Basic examples, we'll use the ***SqlConnection*** and ***SqlCommand*** classes to communicate with the SQL Server Express sample database named "Planets". These are the special classes for talking to any version of Microsoft SQL Server. We'll work with the data further in two different ways

1. Using the **SqlDataReader** class.   
     
   The SqlDataReader class allows you a lot of programming control since you can step through each data row yourself and choose what to do with the values you get back.
2. Using the **SqlDataAdapter** and **DataSet** classes.   
     
   Datasets are useful if you wish to have the data rows automatically displayed in a forms control such as a datagrid. This approach requires quite a few lines of code to get the data from the database, but saves a lot of trouble in displaying that same data - because smart controls like the DataGridView understand how to hook themselves up to a dataset.

##### **Example Program 12**

The following program connects to a SQL Server Express database and sends it a SQL query. It then gets the results back, steps through each row and writes each PlanetName value on a new line in a Label control.

| Code for program 12a (SQL Express Version – see disk example 12b for Microsoft Access version) |
| --- |
| Imports System  Imports System.Windows.Forms  Imports System.Data  Imports System.Data.SqlClient ' Namespace for working with SQL Server data  Class SimpleDataAccess  Inherits Form  Public Sub New()  ' Set the window title  Me.Text = "A simple databasing program"  ' Determine the physical path to the PLANETS sample database  Dim dbLocation As String = \_  System.IO.Path.GetFullPath("../../../database/SqlServer/planets.mdf")  ' Add a label that fills the form  Dim label1 As Label = New Label()  label1.Dock = DockStyle.Fill  Me.Controls.Add(label1)  ' Connect to the SQL Server database  Dim connection1 As SqlConnection = New SqlConnection( \_  "data source=.\SQLEXPRESS;" \_  & "User Instance=true;Integrated Security=SSPI;AttachDBFilename=" \_  & dbLocation)  connection1.Open()  ' Talk to the database - ask it for data on planets  Dim sql As String = "SELECT \* FROM PLANET"  Dim command1 As SqlCommand = New SqlCommand(sql, connection1)  Dim dataReader1 As SqlDataReader = command1.ExecuteReader()  ' Loop through the records returned and add each planet name to the label  While dataReader1.Read()  label1.Text = \_  label1.Text & dataReader1("PlanetName").ToString() & Environment.NewLine  End While  ' Clean up  dataReader1.Close()  connection1.Close()  End Sub  Shared Sub Main()  ' Start a new instance of a forms application, using our class above  Application.Run(New SimpleDataAccess())  End Sub  End Class |

##### 

##### **Example Program 13**

In this next program we want to display *several columns* of data, which would be too messy in a Label - so we use a DataGridView.

We execute the same query as the previous program but this time put the results into a DataSet. We then hook the DataGridView to the DataSet and it automatically displays all the data.

Hooking up some invisible back-end data to a visual control is referred to as **data binding**.

| Code for program 13a (SQL Express version – see disk example 13b for Microsoft Access version) |
| --- |
| Imports System.Windows.Forms  Imports System.Data  Imports System.Data.SqlClient ' Namespace for working with SQL Server databases  Class DataInGrid  Inherits Form  Public Sub New()  ' Set the window title  Me.Text = "One-Way Database Grid Binding"  ' Determine the physical path to the PLANETS sample database  Dim dbLocation As String = \_  System.IO.Path.GetFullPath("../../../database/SqlServer/planets.mdf")  ' Add a DataGridView to the form  Dim DataGridView1 As DataGridView = New DataGridView()  DataGridView1.Width = Me.Width  DataGridView1.Height = 250  DataGridView1.DataMember = "Table"  DataGridView1.Dock = DockStyle.Fill  Me.Controls.Add(DataGridView1)  ' Connect to the SQL Server database  Dim connection1 As SqlConnection = New SqlConnection( \_  "data source=.\SQLEXPRESS;" \_  & "User Instance=true;Integrated Security=SSPI;AttachDBFilename=" \_  & dbLocation)  connection1.Open()  ' The DataSet will hold the data in memory (in structures called DataTables)  Dim dataSet1 As DataSet = New DataSet()  ' The DataAdapter will be the bridge between the database and the dataset  Dim sqlDataAdapter1 As SqlDataAdapter = New SqlDataAdapter()  ' Tell the DataAdapter what we want it to fetch, and where from  sqlDataAdapter1.SelectCommand = New SqlCommand("SELECT \* FROM PLANET", connection1)  ' Fill the in-memory DataSet with the data now  sqlDataAdapter1.Fill(dataSet1)  ' Hook the DataGridView (the visual grid) to the in-memory data  DataGridView1.DataSource = dataSet1  ' Close the database connection  connection1.Close()  End Sub  Shared Sub Main()  ' Start a new instance of a forms application, using our class above  Application.Run(New DataInGrid())  End Sub  End Class |

##### 

##### **Example Program 14**

Displaying data in a DataGridView is okay, but you’ll notice that if you change the data, it does not get saved back into the database. So let’s modify the approach to allow "**two-way data binding**".

We’ll cheat a little bit here (hey, it’s called "increasing our productivity") by not writing our own UPDATE and INSERT SQL commands – the System.Data namespace has a smart little class called CommandBuilder that can figure out how to write them itself and handles them behind the scenes.

| Code for program 14a (SQL Express version – see disk example 14b for Microsoft Access version) |
| --- |
| Imports System.Windows.Forms  Imports System.Data  Imports System.Data.SqlClient ' Namespace for working with SQL Server databases  Class PlanetsForm  Inherits Form  ' Declare some objects we'll be talking to from different methods  Private dg As DataGridView  Private da As SqlDataAdapter  Public Sub New()  ' This is the "constructor" method for the PlanetsForm class  ' Set the window title  Me.Text = "Two-way Database Grid Binding"  ' Determine the physical path to the PLANETS sample database  Dim dbLocation As String = \_  System.IO.Path.GetFullPath("../../../database/SqlServer/planets.mdf")  ' Prepare to connect to the SQL Server database  Dim connectionString As String = "data source=.\SQLEXPRESS;" \_  & "User Instance=true;Integrated Security=SSPI;AttachDBFilename=" \_  & dbLocation  ' Add a "Save" button to the form  Dim btnSave As Button = New Button()  btnSave.Text = "Save"  AddHandler btnSave.Click, AddressOf BtnSave\_Click  btnSave.Dock = DockStyle.Top  Me.Controls.Add(btnSave)  ' Add a DataGridView to the form  dg = New DataGridView()  dg.Width = Me.Width  dg.Height = 250  dg.Dock = DockStyle.Fill  Me.Controls.Add(dg)  ' Instantiate a few objects that are smart with data and use them  ' together to "bind" the DataGridView to the back-end data we want  ' DataAdapter will act as bridge between database and in-memory DataTable  da = New SqlDataAdapter("SELECT \* FROM PLANET", connectionString)  ' CommandBuilder will handle UPDATE and INSERT automatically  Dim cb As SqlCommandBuilder = New SqlCommandBuilder(da)  ' DataTable will keep track, in memory, of changes  Dim dt As DataTable = New DataTable()  ' Fill the DataTable with the data now  da.Fill(dt)  ' Link the DataTable to the DataGridView now  dg.DataSource = dt  End Sub  Shared Sub Main()  ' Start a new instance of a forms application, using our class above  Application.Run(New PlanetsForm())  End Sub  Private Sub BtnSave\_Click(ByVal sender As Object, ByVal e As System.EventArgs)  ' Since the save button was clicked, update the database with  ' any changes made to the DataGridView's underlying source, which  ' in this case is a DataTable.  da.Update(DirectCast(dg.DataSource, DataTable))  MessageBox.Show("Data has been saved", "For your information", MessageBoxButtons.OK)  End Sub  End Class |

##### 

*Try modifying some values and entering new ones. Click "Save" and then close the form. If you re-run the program you will see that the data really has been updated/inserted in the database*.

##### **What If I’m Not Using SQL Server Express?**

Those of you who do not have a version of Microsoft SQL Server installed, and have a different database you wish to talk to, will need to make a few small adjustments.

First of all, the connection string describing the database location, type, etc. must change.

A connection string for SQL Server Express may look like this:

Dim connectionString As String = \_

"Integrated Security=SSPI;Persist Security Info=False; Initial Catalog=***Northwind***;Data Source=***localhost***"

or like this, if you’re connecting directly to the database file (as in this book’s examples):

Dim connectionString As String = \_

"data source=.\SQLEXPRESS;Integrated Security=SSPI;

AttachDBFilename=***c:\Visual Basic4#KIDS\examples\database\SqlServer\planets.mdf***;

User Instance=true"

*(Because of space we’re wrapping this around to a few lines, but in this format the portion in quotes actually needs to be on one line)*

Aconnection string for Microsoft Access could look like this:

Dim connectionString As String = \_

"Provider=Microsoft.Jet.OLEDB.4.0; Data Source c:\Visual Basic4#KIDS\examples\database\Access\planets.mdb;"  
  
*(as explained higher above, you would need to write this on one line)*

Other databases will each have a particular format. You may find examples in the Visual Basic Express help documentation or in the documentation that came with your database.

Beyond changing the connection string, you then also change the "**Sql**" classes to "**OleDb**" classes.

Start by including **System.Data.OleDb** namespace instead of the **System.Data. SqlClient** namespace. This contains classes that can work with a variety of databases.

Imports System.Data.OleDb

Then, swap the classes you use to work with data, as follows:

| **SQL Server** | **General Databases** |
| --- | --- |
| SqlCommand | OleDbCommand |
| SqlCommandBuilder | OleDbCommandBuilder |
| SqlDataAdapter | OleDbDataAdapter |
| SqlConnection | OleDbConnection |

### **System.Xml**

##### **Introduction**

The classes in System.Xml help you to work with XML data in various ways. Common tasks include:

* opening an XML document
* reading a piece of XML to get some specific values out
* writing an XML file to disk

##### **A Brief Introduction to XML**

XML (Extensible Markup Language) is everywhere these days and you're most likely to have heard about it already. XML is a great example of a language that both people and computers can understand. While some computer systems work with data that seems to be a garbled mess to humans, XML is written in plain text and can be read by the average country bumpkin.

You might, for example, put together an XML document like the following to hold some geographical data on disk:

| <?xml version="1.0" encoding="utf-8" ?> | | | | |  |  | Every XML document has this first line simply so programs know there is XML coming |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | |  |  |
| <Earth> |  | | | |  | The outermost block |
|  |  |  | | |  |  |
|  | <Continent> |  | | |  | A nested (indented) block with information about the South American continent |
|  |  | South America | | |
|  |  | <Country capital="Santiago"> | | |
|  |  |  | | Chile |
|  |  | </Country> | |  |
|  |  | <Country capital="Buenos Aires"> | | |
|  |  |  | | Argentina |
|  |  | </Country> | |  |
|  | </Continent> |  | |  |
|  |  |  | | |  |  |
|  | <Continent> |  | | |  |  |
|  |  | Asia | | |  |  |
|  |  | <Country capital="Baghdad"> | | |  | More deeply-nested blocks with information about two *countries* on the Asian continent |
|  |  |  | Iraq | |  |
|  |  | </Country> | | |  |
|  |  |  | | |  |
|  |  | <Country capital="New Delhi"> | | |  |
|  |  |  | India | |  |
|  |  | </Country> | | |  |
|  |  |  | | |  |  |
|  | </Continent> |  | | |  |  |
| </Earth> |  |  | | |  |  |
|  |  |  | | |  |  |

It's a lot like HTML, but you're free to make up *your own tag names* in XML.

Let's talk about two terms you'll need to understand if you're going to read further about XML.

##### **Elements**

##### XML data is held inside **elements**. An element has a name and usually holds a value. In the example below, the element "Country" has the value "Argentina".

| element name | element *value* |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| <Country> | Argentina | </Country> |  |

As you can see above, an element is represented with an *opening tag* and a *closing tag.* The closing tag must be named *exactly* like the opening tag, but must include a forward slash "/".

If the element has no value, it is legal to rather use just one tag and close it immediately.

| element name |
| --- |
|  |
| <Country /> |

##### **Attributes**

But we may want to describe various special things about an element. We may, for example, want to indicate that a country has a capital city - so, as an example, we may create an *attribute* of our element called "capital". In the example below, the "capital" attribute of the "Country" element "Argentina" is equal to "Buenos Aires".

| element name |  | attribute name |  | attribute *value* |  | element *value* |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |
| <Country |  | capital | = | "Buenos Aires" | > | Argentina | </Country> |  |

All this is quite easy for us humans to read and, since there's a clear structure to it, you can imagine that computers can easily be taught to read it too. "Computer, walk through this document, when you get to a "<" you know you're about to read an *element name*. When you reach the next ">" then you know to look for the *element's value* ... and so on.

##### 

##### **Example Program 15**

The following program reads some geographical data in from an XML file and displays it on a form.

It uses three classes from the System.Xml namespace:

* XmlDocument (creates an object that can load XML data so we can work with it)
* XmlNodeList (useful for holding the list of elements we read from the file)
* XmlNode (holds one XML element)

It allows the user to type in something called an **xPath expression** to say what elements to get from the xml file. An xPath expression such as //earth/continent/country means "find all elements named "country" wherever they appear beneath an element named "continent" appearing under an element named "earth".

| Code for program 15 |
| --- |
| Imports System  Imports System.Windows.Forms  Imports System.Drawing  Imports System.Xml ' Namespace for working with XML generally  Imports System.Xml.XPath ' Namespace for working with XML "paths"  Class XmlRetriever  Inherits Form  Private comboBox1 As ComboBox  Private button1 As Button  Private listBox1 As ListBox  Private richTextBox1 As RichTextBox  Private xmlDoc As XmlDocument  ' Constructor method for our class  Public Sub New()  ' Set the window title and height  Me.Text = "XML Retrieval"  Me.Size = New Size(400, 400)  ' Load an XML file from disk into an XmlDocument object in memory  xmlDoc = New XmlDocument()  xmlDoc.Load("../../data/earthData.xml")  ' Prepare a large text box to show the data in  richTextBox1 = New RichTextBox()  richTextBox1.Dock = DockStyle.Top  richTextBox1.AcceptsTab = True  richTextBox1.Height = 180  richTextBox1.[ReadOnly] = True  richTextBox1.BackColor = Color.Silver  ' Put the XML into the text box and add the textbox to the form  richTextBox1.Text = xmlDoc.OuterXml  Me.Controls.Add(richTextBox1)  ' Add a combobox control that allows the user to either type  ' an Xpath expression or to choose an existing one. Add some  ' as examples to help the user learn to write XPath expressions.  ' (A combo box is a combination of a textbox and a listbox)  comboBox1 = New ComboBox()  comboBox1.Location = New Point(0, 200)  comboBox1.Width = 300  comboBox1.Items.Add("//Earth/Continent")  comboBox1.Items.Add("//Earth/Continent/Country")  comboBox1.Items.Add("//Earth/Continent/Country[@capital='Nairobi']")  comboBox1.SelectedIndex = 0  Me.Controls.Add(comboBox1)  ' Add a button that will cause the matching items to be displayed  button1 = New Button()  button1.Text = "Get data"  button1.Location = New Point(310, 200)  AddHandler button1.Click, AddressOf Button1\_Click  Me.Controls.Add(button1)  ' Add a listbox to display the items returned  listBox1 = New ListBox()  listBox1.Dock = DockStyle.Bottom  listBox1.Location = New Point(10, 10)  Me.Controls.Add(listBox1)  End Sub  Shared Sub Main()  ' Start a new instance of a forms application, using our class above  Application.Run(New XmlRetriever())  End Sub  ' An eventhandler that will run when the button is clicked  Private Sub Button1\_Click(ByVal sender As Object, ByVal e As EventArgs)  Dim xmlNodes As XmlNodeList  Dim xmlElement As XmlNode  Dim elementValue As String  ' Use a try-catch structure to catch and handle any XPath errors  ' neatly instead of just letting the program crash  Try  ' From the XML document, select the items specified by the  ' XPath query chosen in the combobox  xmlNodes = xmlDoc.SelectNodes(comboBox1.Text)  listBox1.Items.Clear()  ' Loop through all the macthing items that were found,  ' adding each item to the listbox  For i As Integer = 0 To xmlNodes.Count - 1  xmlElement = xmlNodes(i)  If xmlElement.HasChildNodes Then  elementValue = xmlElement.FirstChild.Value.Trim()  listBox1.Items.Add(elementValue)  End If  Next  Catch ex As XPathException  ' If an XPath error was encountered, tell the user neatly and carry on  MessageBox.Show("No matching data was found. Try another xPath expression.", \_  "Error Encountered. " + ex.Message)  End Try  End Sub  End Class |

The program shows the raw xml file in a "RichTextBox" control. Then, below the xPath expression which the user can modify, the resulting element values are shown. So once you get this program running, try changing the expression and clicking the "Get data" button.

##### 

**About the Book**

Visual Basic is both a powerful and easy to use programming language. It presents a great opportunity for the new generation of developers to start out with a language that is highly respected in the modern workplace.

This text introduces object-oriented programming to the young developer (core target age is 12-16) in a lightweight fashion, allowing them to get started with real programs in a Windows environment.

**About the Author**

|  | Martin Dreyer is an ex-high school teacher who now heads a team of software developers in South Africa.  His formal qualifications are a Higher Diploma in Education : Physical Science and a Bachelor of Science Degree : Computer Science and Information Systems. |
| --- | --- |