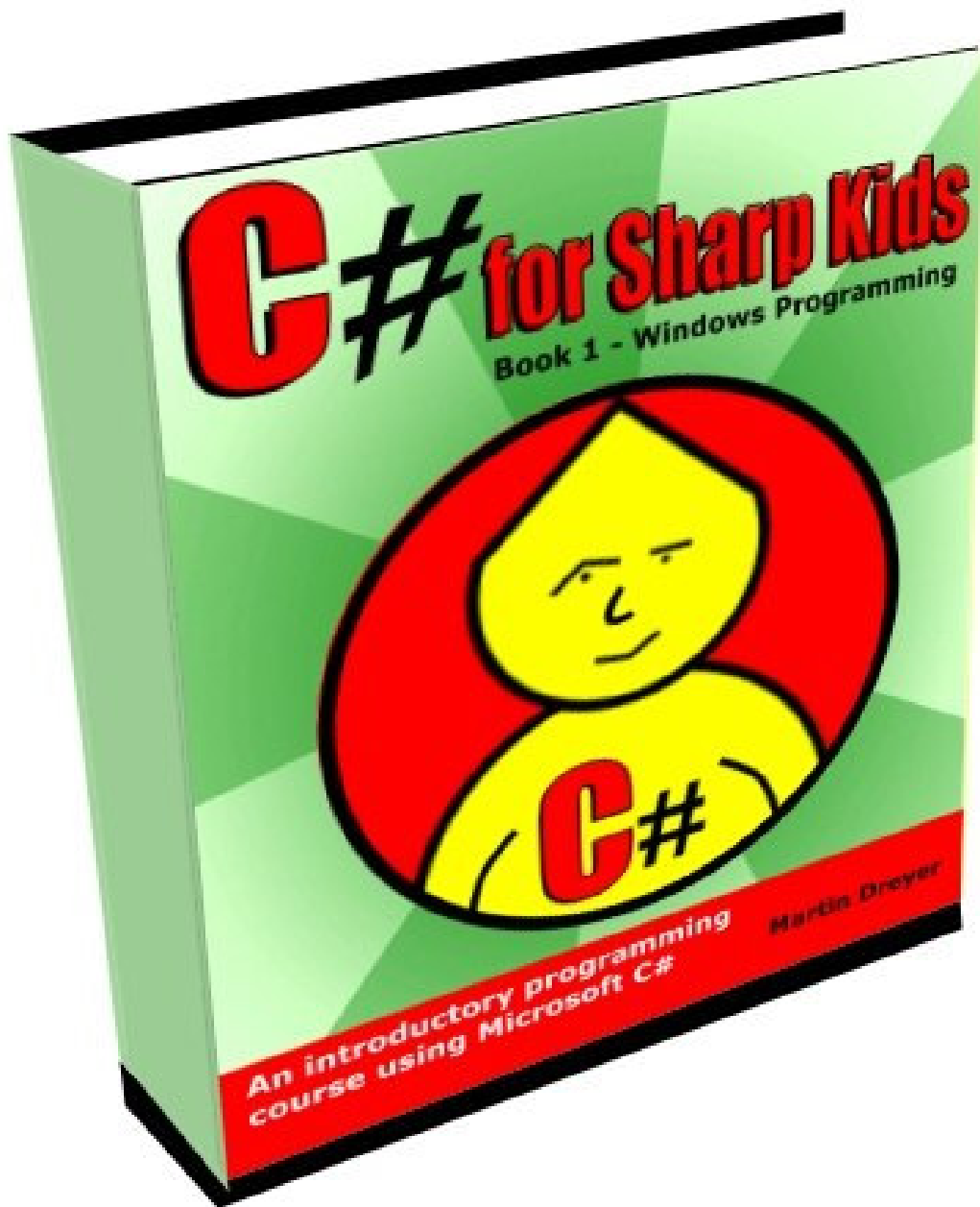
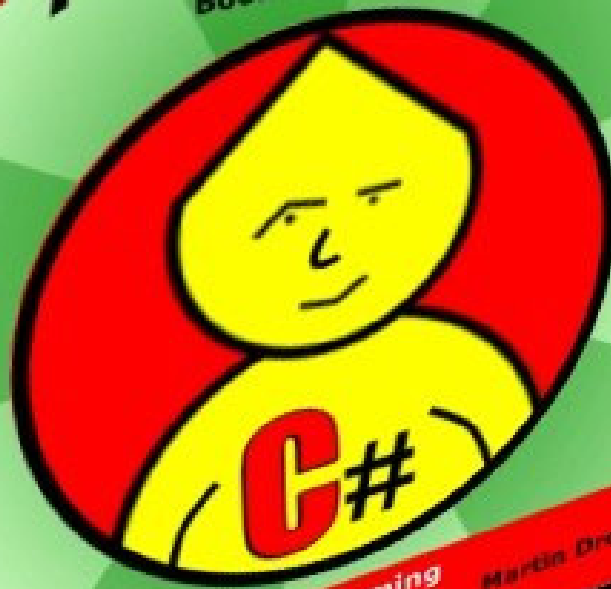




C# for Sharp Kids

Book 1 - Windows Programming



An introductory programming
course using Microsoft C#

Martin Dreyer

Table of Contents

Part IV – Programming with the .NET Framework	3
System.Console	7
System.Windows.Forms	10
System.Drawing.....	25
System.Data	35
System.Xml	48

Part IV – Programming with the .NET Framework

Having C# 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. C# 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 C# 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 C# 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

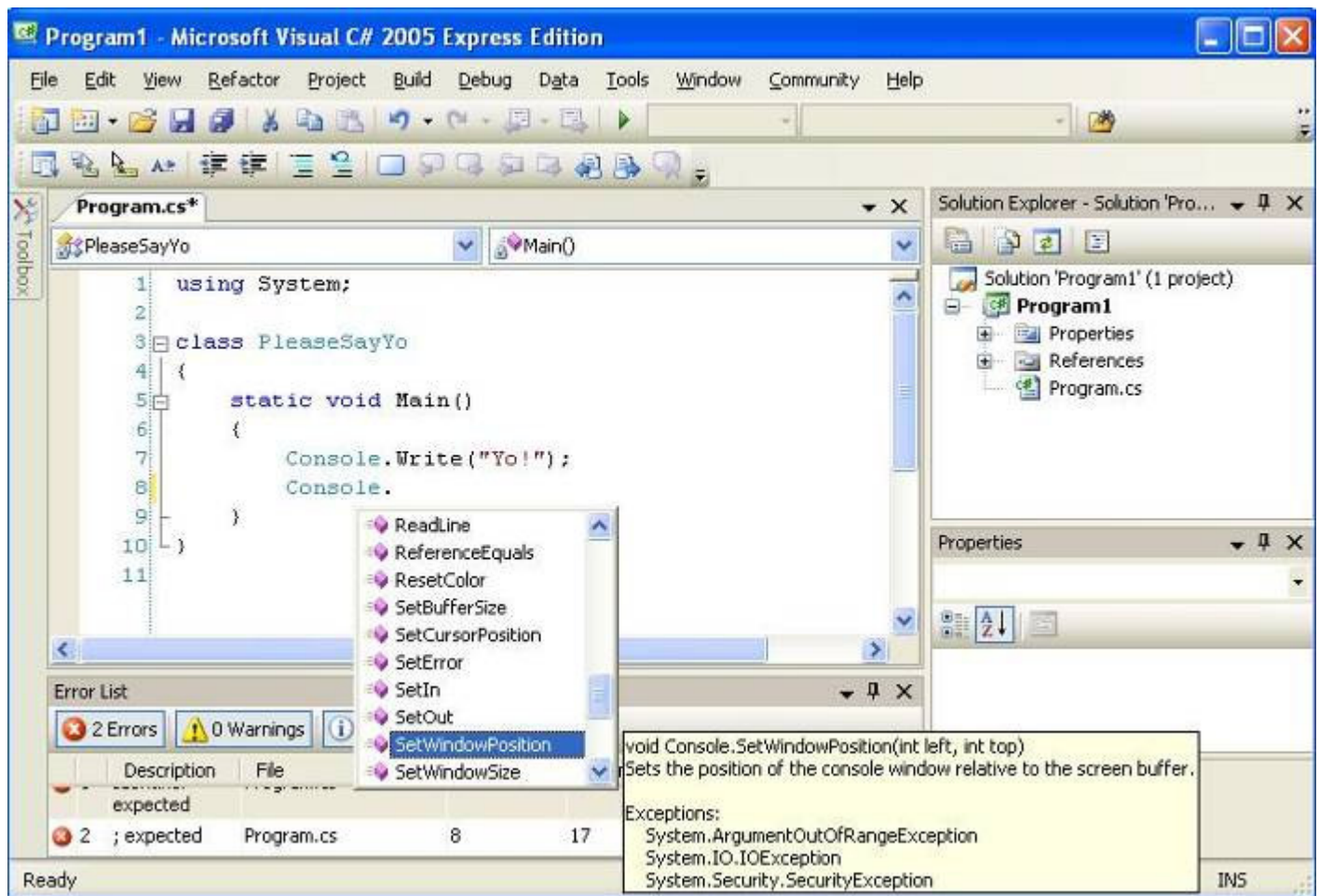
In the following chapters, whenever you see a block with the words "code for program ..." you can run that code using Visual C# Express. Although most of these programs are short, they are all real working programs. Here are the steps you will normally follow:

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

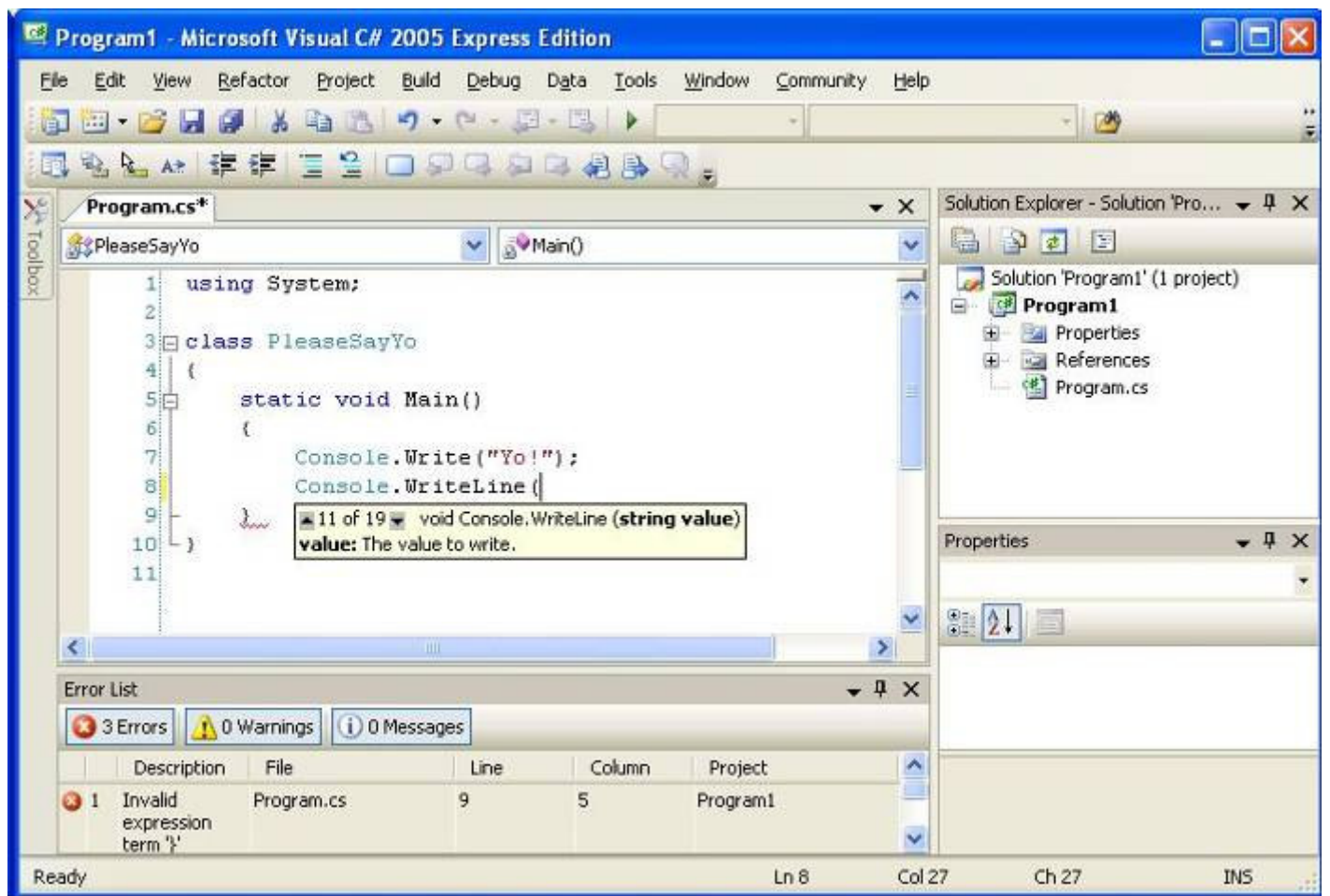
How to Change and Extend the Examples

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 C# Express's help.
- While programming in Visual C# 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 C# Express shows that there are 19 different ways you can call WriteLine. In the picture below, we pressed the down arrow until we reached the 11th 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 the 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

٤. **Console.Write** – writes some characters to the screen without starting a new line

Example Program 1

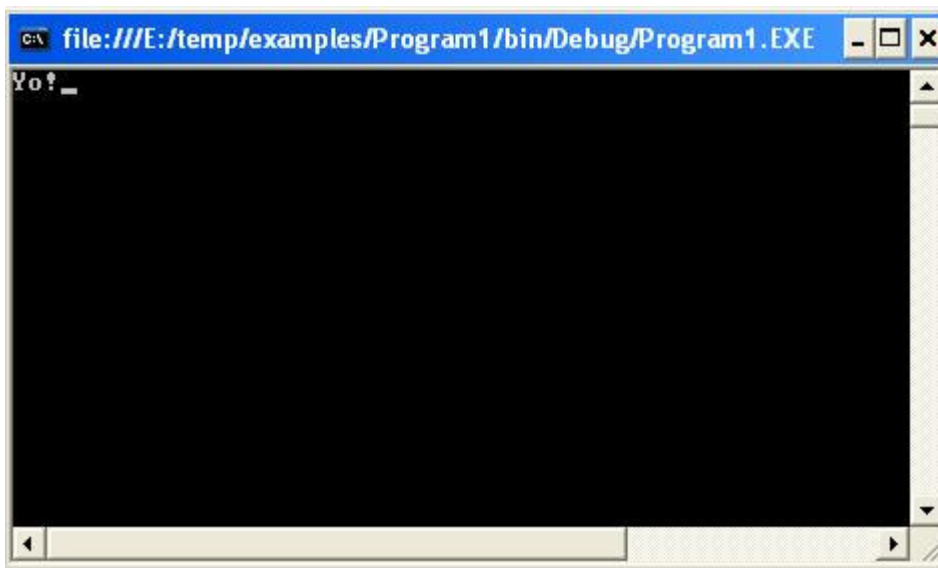
The following program simply writes the word "Yo!" to the screen and then waits for the ENTER key to be pressed.

Code for program 1

```
using System;

class PleaseSayYo
{
    static void Main()
    {
        // Write a word to the screen
        Console.Write("Yo!");

        // Wait until enter is pressed
        Console.ReadLine();
    }
}
```



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

```
using System;

class WriteAFewLines
{
    static void Main()
    {
        Console.WriteLine("Type in your favourite word : ");

        // Store, in a string variable, the word that the user entered
        string favouriteWord = Console.ReadLine();

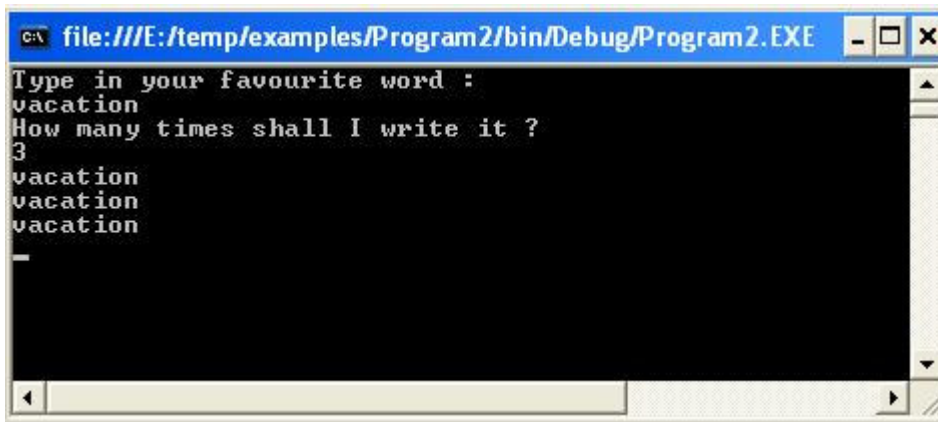
        Console.WriteLine("How many times shall I write it? ");

        // Store, in an integer variable, the number they type in
        // (And they'll get an error if they don't type a true number)
        int numberOfTimes = Convert.ToInt32(Console.ReadLine());

        // Write the word out as many times as was requested

        for (int i = 0; i < numberOfTimes; i++)
        {
            Console.WriteLine(favouriteWord);
        }

        // Wait until ENTER is pressed
        Console.ReadLine();
    }
}
```



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 C# 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 C# Express

To play with the following examples in C# Express, you can select File -> Open Project and open the book's sample .csproj 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, C# Express puts down some **.cs** files for you (named **Form1.cs** and **Program.cs**), and inserts some C# 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.cs and
- replace the code in Program.cs 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.

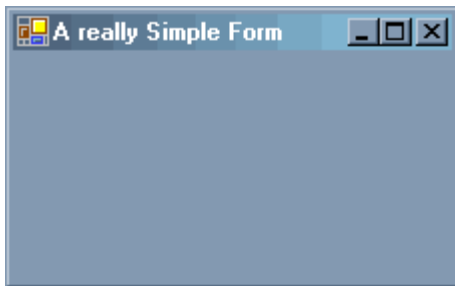
Code for program 3

```
using System.Windows.Forms;

class SimpleWindowsForm : Form
{
```

```
// The constructor method for our class
public SimpleWindowsForm()
{
    // Set the window title
    this.Text = "A really Simple Form";
}

static void Main()
{
    // Start a new instance of a forms application, using our class above
    Application.Run(new SimpleWindowsForm());
}
}
```



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

```
using System.Windows.Forms;

class SimpleWindowsFormWithButton : Form
{
    Button button1;

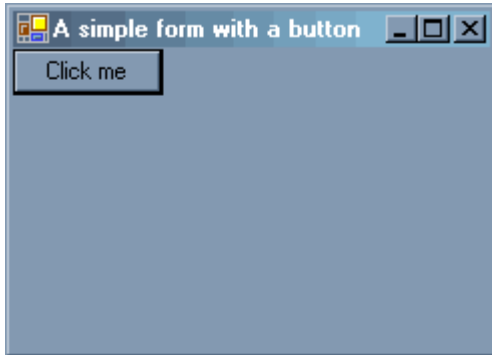
    // The constructor method for our class
    public SimpleWindowsFormWithButton()
    {
        // Set the window title
        this.Text = "A simple form with a button";

        // 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";
        this.Controls.Add(button1);
    }

    static void Main()
```

```
{  
    // Start a new instance of a forms application, using our class above  
    Application.Run(new SimpleWindowsFormWithButton());  
}
```



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

```
using System;  
using System.Windows.Forms;  
using System.Drawing;  
  
class FormWithWorkingButton : Form  
{  
    // The constructor method for our class  
    public FormWithWorkingButton()  
    {  
        // Set the window title  
        this.Text = "Simple form with working button";  
  
        // Add a button and hook it up to an event handler method  
  
        Button button1 = new Button();  
        button1.Text = "Click me";  
        button1.Click += new System.EventHandler(Button1_Click);  
        this.Controls.Add(button1);  
    }  
}
```

```

    }

    static void Main()
    {
        // Start a new instance of a forms application, using our class above
        Application.Run(new FormWithWorkingButton());
    }

    // An event handler that will be run when the button is clicked
    void Button1_Click(object sender, EventArgs e)
    {
        // Change the window title
        this.Text = "Button was clicked";
    }
}

```



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

```

using System.Windows.Forms;
using System.Drawing;

class MyForm : Form
{
    // Declare a listbox outside the methods, since we'll
    // want to talk to it from more than one method
    ListBox listBox1;

    // The constructor method for our class
    public MyForm()
    {

```

```

        // Create a picture box, put a picture in it and add it to this form

        PictureBox pictureBox1 = new PictureBox();
        pictureBox1.SizeMode = PictureBoxSizeMode.StretchImage;
        Bitmap image1 = new Bitmap("../..../images/seashore.jpg");
        pictureBox1.ClientSize = new Size(this.Width, 100);
        pictureBox1.Image = (Image)image1;
        this.Controls.Add(pictureBox1);

        // Create a button object, set some of it's properties and add it to the form

        Button button1 = new System.Windows.Forms.Button();
        button1.Location = new System.Drawing.Point(10, 120);
        button1.Text = "Click me";
        button1.Click += new System.EventHandler(button1_Click);
        this.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;

        this.Controls.Add(listBox1);
    }

    // An event handler method that will run when the button is clicked
    void button1_Click(object sender, System.EventArgs e)
    {
        // Show a message indicating which item in the listbox is selected
        MessageBox.Show(this, "You have selected " + listBox1.SelectedItem,
"Notification", MessageBoxButtons.OK);
    }

    static void Main()
    {
        // Start a new instance of a forms application, using our class above
        Application.Run(new MyForm());
    }
}

```



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

```
using System;
using System.Windows.Forms;
using System.Drawing;
using System.Data;
using System.Xml;

class FormWithManyControls : Form
{
    TreeView treeView1;
    Panel panel1;
    CheckBox checkBox1, checkBox2;
    RadioButton radioButton1, radioButton2;
    ListBox listBox1;

    // The constructor method for our class
    public FormWithManyControls()
```

```

{
    // Set window title and size

    this.Text = "A form dealing with many types of controls";
    this.Width = 700;
    this.Height = 500;

    // Add a tree view as a kind of menu

    treeView1 = new TreeView();
    treeView1.BackColor = Color.BurlyWood;
    treeView1.Dock = DockStyle.Left;
    treeView1.AfterSelect +=
        new System.Windows.Forms.TreeViewEventHandler(treeView1_AfterSelect);

    TreeNode tn = 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);

    this.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 = this.Width - treeView1.Width;

    this.Controls.Add(panel1);
}

// An eventhandler that runs when something in the treeview is selected
private void treeView1_AfterSelect
    (object sender, System.Windows.Forms.TreeViewEventArgs e)
{
    // Do something appropriate for whichever node was clicked

    if (e.Node.Text == "[Wipe clean]")
    {
        // Wipe the panel clean of any controls
        panel1.Controls.Clear();
    }
}

```

```

}
else if (e.Node.Text == "Button")
{
    // Add a button to the panel

    Button button1 = new Button();
    button1.Text = "I'm a button";
    button1.Location = new Point(150, 80);
    button1.Click += new EventHandler(button1_Click);

    panel1.Controls.Add(button1);
}
else if (e.Node.Text == "Label")
{
    // Add a label to the panel

    Label label1 = 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;
    label1.Click += new EventHandler(label1_Click);

    panel1.Controls.Add(label1);
}
else if (e.Node.Text == "CheckBox")
{
    // Add some checkboxes to the panel

    checkBox1 = new CheckBox();
    checkBox1.Text = "I am sharp";
    checkBox1.Location = new Point(10, 50);
    checkBox1.CheckedChanged +=
        new EventHandler(CheckBox_CheckedChanged);
    panel1.Controls.Add(checkBox1);

    checkBox2 = new CheckBox();
    checkBox2.Text = "I am modest";
    checkBox2.Location = new Point(10, 70);
    checkBox2.CheckedChanged +=
        new EventHandler(CheckBox_CheckedChanged);
    panel1.Controls.Add(checkBox2);
}
else if (e.Node.Text == "RadioButton")
{
    // Add some radio buttons to the panel

    radiobutton1 = new RadioButton();
    radiobutton1.Text = "I am sharp";
    radiobutton1.Location = new Point(370, 210);
    radiobutton1.CheckedChanged +=
        new EventHandler(RadioButton_CheckedChanged);

    panel1.Controls.Add(radiobutton1);
}

```

```

        radioButton2 = new RadioButton();
        radioButton2.Text = "I am dim-witted";
        radioButton2.Location = new Point(370, 230);
        radioButton1.CheckedChanged +=
            new EventHandler(RadioButton_CheckedChanged);

        panell1.Controls.Add(radioButton2);
    }
    else if (e.Node.Text == "ListBox")
    {
        // 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);
        listBox1.SelectedIndexChanged +=
            new EventHandler(listBox1_SelectedIndexChanged);

        panell1.Controls.Add(listBox1);
    }
    else if (e.Node.Text == "TextBox")
    {
        // Add a text box to the panel

        TextBox TextBox1 = new TextBox();
        TextBox1.Text = "You can type here";
        TextBox1.Location = new Point(10, 110);

        panell1.Controls.Add(TextBox1);
    }
    else if (e.Node.Text == "DataGridView")
    {
        // Add a datagrid, populated with data from an xml file, to the panel

        DataSet dataSet1 = new DataSet("A sample DataSet");
        dataSet1.ReadXml("../data/grades.xml");

        DataGridView dataGridView1 = new DataGridView();
        dataGridView1.Width = (panell1.Width / 2) - 10;
        dataGridView1.Height = 150;
        dataGridView1.Location =
            new Point(2, panell1.Height - dataGridView1.Height - 5);

        dataGridView1.DataSource = dataSet1;
        dataGridView1.DataMember = "subject";
        panell1.Controls.Add(dataGridView1);
    }
    else if (e.Node.Text == "TabControl")
    {
        // Add a tab control to the panel
        // and add some content to each tab page

        TabControl tabControl1 = new TabControl();

```

```

        tabControl1.Location = new Point(10, 150);
        tabControl1.Size = new Size(165, 146);

        TabPage tabPage1 = new TabPage("Robynne");
        PictureBox pictureBox1 = new PictureBox();
        pictureBox1.Image = new Bitmap("../..//images/robynne.jpg");
        pictureBox1.Size = new Size(160, 120);
        tabPage1.Controls.Add(pictureBox1);
        tabControl1.TabPages.Add(tabPage1);

        TabPage tabPage2 = new TabPage("Her Dog");
        PictureBox pictureBox2 = new PictureBox();
        pictureBox2.Image = new Bitmap("../..//images/chocolate.jpg");
        pictureBox2.Size = new Size(160, 120);
        tabPage2.Controls.Add(pictureBox2);
        tabControl1.TabPages.Add(tabPage2);

        TabPage tabPage3 = new TabPage("Info");
        tabPage3.BackColor = Color.White;
        Label label1 = new Label();
        label1.Text =
            "Robynne lives in Cape Town, South Africa.\n\n" +
            "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);
    }
    else if (e.Node.Text == "PictureBox")
    {
        // Add a picture to the panel

        PictureBox pictureBox1 = 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);
    }
    else if (e.Node.Text == "RichTextBox")
    {
        // Add a box for typing rich text in
        // Load some data from an XML file into it

        RichTextBox richTextBox1 = new RichTextBox();
        richTextBox1.LoadFile("../..//data/grades.xml",
            RichTextBoxStreamType.PlainText);
        richTextBox1.WordWrap = false;
        richTextBox1.BorderStyle = BorderStyle.Fixed3D;
        richTextBox1.BackColor = Color.Beige;
        richTextBox1.Size = new Size((panel1.Width / 2) - 10, 150);
        richTextBox1.Location =
            new Point((panel1.Width / 2) + 10,

```

```

        panel1.Height = richTextBox1.Height - 5);

        panel1.Controls.Add(richTextBox1);
    }
    else if (e.Node.Text == "MainMenu")
    {
        // Add a classic "menu" (appears at the top of the window)
        MainMenu mainMenu1 = new MainMenu();

        MenuItem menuItem1 = new MenuItem("File");
        menuItem1.MenuItems.Add("Exit",
            new EventHandler(mainMenu1_Exit_Select));
        mainMenu1.MenuItems.Add(menuItem1);

        MenuItem menuItem2 = new MenuItem("Background");
        menuItem2.MenuItems.Add("Choose",
            new EventHandler(mainMenu1_ColorOwn_Select));
        menuItem2.MenuItems.Add("White",
            new EventHandler(mainMenu1_ColorWhite_Select));
        mainMenu1.MenuItems.Add(menuItem2);

        this.Menu = mainMenu1;

        MessageBox.Show("A main menu has been added at the top " +
            "left of the window. Try it out after clicking OK.");
    }
    else if (e.Node.Text == "ToolBar")
    {
        // Add a shortcuts toolbar to the panel

        ToolBar toolBar1 = new ToolBar();
        ImageList imageList1 = 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;

        ToolBarButton toolBarButton1 = new ToolBarButton("Open");
        toolBarButton1.ImageIndex = 0;
        toolBar1.Buttons.Add(toolBarButton1);

        ToolBarButton toolBarButton2 = new ToolBarButton("Close");
        toolBarButton2.ImageIndex = 1;
        toolBar1.Buttons.Add(toolBarButton2);

        ToolBarButton toolBarButton3 = new ToolBarButton("Huh");
        toolBarButton3.ImageIndex = 3;
        toolBar1.Buttons.Add(toolBarButton3);

        toolBar1.ButtonClick +=
            new ToolBarButtonClickEventHandler(toolBar1_Click);

        panel1.Controls.Add(toolBar1);
    }

```

```

}

/* All other event handlers - for the controls added above */

// An eventhandler that will be run if the label is clicked
void label1_Click(object sender, System.EventArgs e)
{
    MessageBox.Show
        ("Yes, labels can be clicked, although it's not normal to do so.");
}

// An eventhandler that will be run if the button is clicked
void button1_Click(object sender, System.EventArgs e)
{
    MessageBox.Show("Yup, you were supposed to click me");
}

// An eventhandler that will be run if a checkbox is checked/unchecked
void CheckBox_CheckedChanged(object sender, System.EventArgs e)
{
    if (checkBox1.Checked && checkBox2.Checked)
    {
        MessageBox.Show("Good for you");
    }
    else if (checkBox1.Checked)
    {
        MessageBox.Show("It's not good to be sharp without being modest");
    }
    else if (checkBox2.Checked)
    {
        MessageBox.Show("Modesty is good. Pity you're not sharp too.");
    }
    else
    {
        MessageBox.Show("Oh dear, neither sharp nor modest eh?");
    }
}

// An eventhandler that will be run if a radiobutton is clicked
void RadioButton_CheckedChanged(object sender, System.EventArgs e)
{
    if (radiobutton1.Checked)
    {
        MessageBox.Show("Glad to hear it");
    }
    else if (radioButton2.Checked)
    {
        MessageBox.Show("What a shame");
    }
}

// An eventhandler that will be run if an item in the listbox is selected
void listBox1_SelectedIndexChanged(object sender, System.EventArgs e)
{
    if (listBox1.SelectedItem.ToString() == "Green")
    {

```

```

        treeView1.BackColor = Color.LightSeaGreen;
    }
    else if (listBox1.SelectedItem.ToString() == "Beige")
    {
        treeView1.BackColor = Color.Beige;
    }
    else if (listBox1.SelectedItem.ToString() == "White")
    {
        treeView1.BackColor = Color.White;
    }
}

// An eventhandler that will be run if "white" is selected at the menu
void mainMenu1_ColorWhite_Select(object sender, System.EventArgs e)
{
    treeView1.BackColor = Color.White;
}

// An eventhandler that will be run if a color is selected at the menu
void mainMenu1_ColorOwn_Select(object sender, System.EventArgs e)
{
    ColorDialog colorDialog1 = new ColorDialog();
    colorDialog1.Color = treeView1.BackColor;
    colorDialog1.ShowDialog();
    treeView1.BackColor = colorDialog1.Color;
}

// An eventhandler that will be run if the user chooses "exit" at the menu
void mainMenu1_Exit_Select(object sender, System.EventArgs e)
{
    if (
        MessageBox.Show("Are you sure you want to exit?",
            "Exit confirmation", MessageBoxButtons.YesNo)
        == DialogResult.Yes
    )
    {
        this.Dispose();
    }
}

// An eventhandler that will be run if a toolbar shortcut is selected
void toolBar1_Click
(object sender, System.Windows.Forms.ToolBarButtonClickEventArgs e)
{
    if (e.Button.Text == "Open")
    {
        MessageBox.Show("This could have opened a file, for example");
    }
    else if (e.Button.Text == "Close")
    {
        MessageBox.Show("This could have closed a file, for example");
    }
    else if (e.Button.Text == "Huh")
    {
        MessageBox.Show("Huh?");
    }
}

```

```

}

static void Main()
{
    // Start a new instance of a forms application, using our class above
    Application.Run(new FormWithManyControls());
}
}

```

A form dealing with many types of controls

File Background

Controls

- [Wipe clean]
- Label
- Button
- CheckBox
- RadioButton
- ListBox
- TextBox
- TabControl
- DataGridView
- MainMenu
- ToolBar
- PictureBox
- RichTextBox

Open Close Huh

This is a label. Labels are simply used to write text at a point on the screen

☐ I am sharp

☐ I am modest

I'm a button

You can type here

Robynne Her Dog Info



Green
Beige
White

☐ I am sharp

☐ I am dim-witted

	name	grade
▶	Math	63%
	Science	71%
	French	55%
*		

```

<grades>
  <subject>
    <name>Math</name>
    <grade>63%</grade>
  </subject>
  <subject>
    <name>Science</name>
    <grade>71%</grade>
  </subject>

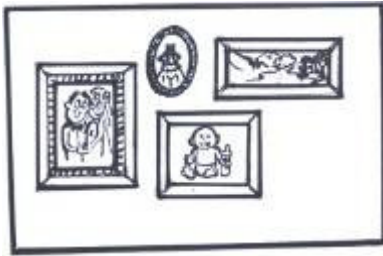
```

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



Bitmap graphics are pictures made up of many dots. For example, photographs and icons can be represented well as bitmaps.

Vector graphics



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

```
using System;
using System.Windows.Forms;
using System.Drawing;

class PictureDisplayer : Form
```

```

{
    Bitmap image1;
    PictureBox pictureBox1;

    // The constructor method for our class
    public PictureDisplay()
    {
        // Set the window title and size

        this.Text = "Picture Displayer";
        this.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

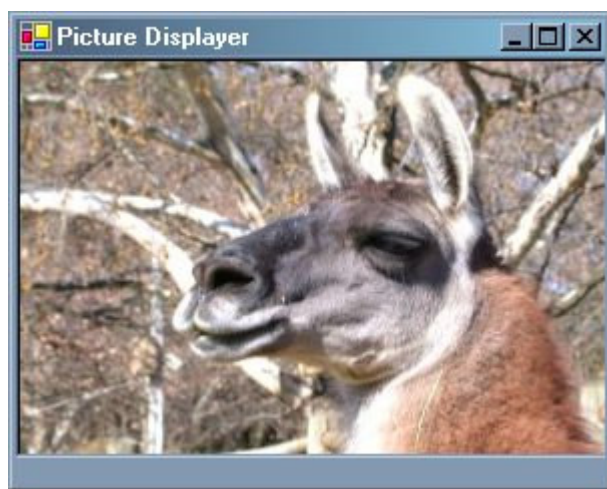
        image1 = new Bitmap(@"../../images/lama.jpg");
        pictureBox1.Image = (Image)image1;

        // Add the box (with it's picture) to the form

        this.Controls.Add(pictureBox1);
    }

    static void Main()
    {
        // Start a new instance of a forms application, using our class above
        Application.Run(new PictureDisplay());
    }
}

```



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

```
using System;
using System.Windows.Forms;
using System.Drawing;

class PictureFlipper : Form
{
    Button button1;
    Bitmap image1;
    PictureBox pictureBox1;

    // The constructor method for our class
    public PictureFlipper()
    {
        // Set the window title and size

        this.Text = "Picture Flipper";
        this.Size = new Size(302, 240);

        // Add a button to the form

        button1 = new Button();
        button1.Text = "Flip";
        button1.Location = new Point(100, 150);
        button1.Click += new System.EventHandler(button1_Click);
        this.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 = (Image)image1;

        // Add the picture box (with its image) to the form

        this.Controls.Add(pictureBox1);
    }

    static void Main()
    {

```

```

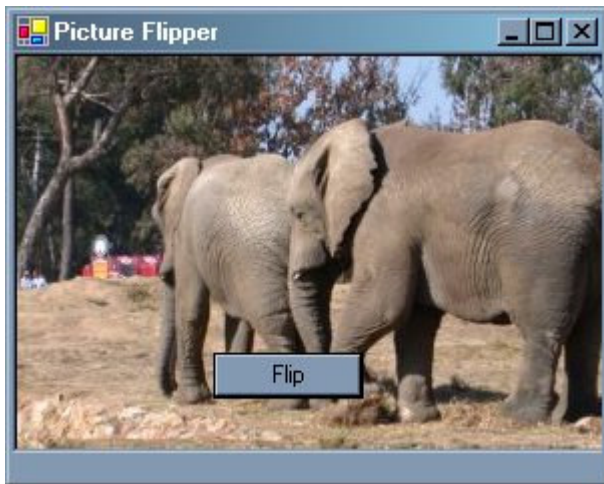
        // Start a new instance of a forms application, using our class above
        Application.Run(new PictureFlipper());
    }

    // An eventhandler that will run when the button is clicked
    void button1_Click(object sender, EventArgs e)
    {
        // Flip the image along the X axis (horizontally)
        image1.RotateFlip(RotateFlipType.RotateNoneFlipX);

        // Re-insert the image into the picture box
        pictureBox1.Image = (Image)image1;

        // Update the window title
        this.Text = "Picture was flipped";
    }
}

```



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 :

```
Pen myGreenPen = new Pen( Color.Green, 3 );
```

- ۲. 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 :

```
SolidBrush myBlueBrush = new SolidBrush( Color.Blue );
```

Example Program 10

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

Code for program 10

```
using System;
using System.Windows.Forms;
using System.Drawing;

class SimpleShapeMaker : Form
{
    // The constructor method for our class
    public SimpleShapeMaker()
    {
        // Set the form background to white

        this.BackColor = Color.White;

        // Add a button, hooked up to an event handler, to the form

        Button button1 = new Button();
        button1.Text = "click me";
        button1.Location = new Point(110, 10);
        button1.BackColor = Color.SteelBlue;
        button1.Click += new System.EventHandler(button1_Click);
        this.Controls.Add(button1);
    }

    // An eventhandler that will run when the button is clicked
    void button1_Click(object o, System.EventArgs e)
    {
        // Execute a specific method we wrote
        DrawSomeShapes();
    }

    // A method that draws a few shapes on the form surface
    void DrawSomeShapes()
    {
        // Prepare a drawing surface for this form
```

```

Graphics g = this.CreateGraphics();

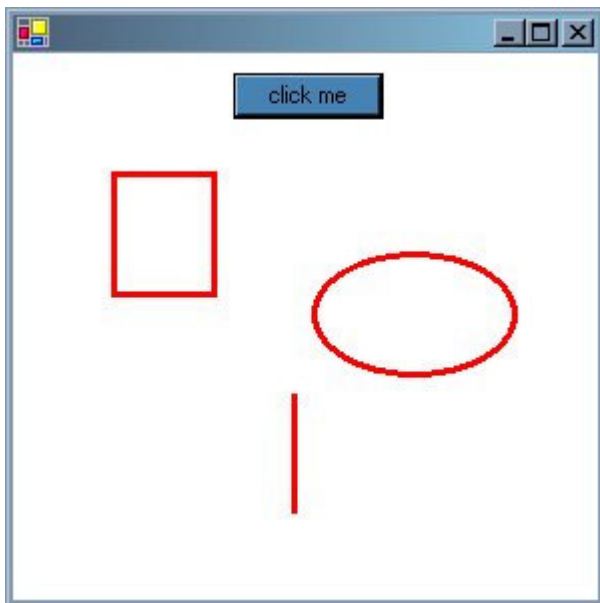
// Prepare a pen that draws red and with a width of 3
Pen redPen = new Pen(Color.Red, 3);

// 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();
}

static void Main()
{
    // Start a new instance of a forms application, using our class above
    Application.Run( new SimpleShapeMaker() );
}
}

```



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

```
using System;
using System.Windows.Forms;
using System.Drawing;

class FunWithTheMouse : Form
{
    // Declare some objects we'll want to talk to from several methods

    PictureBox pictureBox1;
    Label label1;
    Point spotClicked;

    // The constructor method for our class
    public FunWithTheMouse()
    {
        // Set the window size

        this.Size = new Size(640, 480);

        // Load a photo into a picture box and add it to the form

        pictureBox1 = new PictureBox();
        pictureBox1.Image = (Image)new Bitmap(@"../images/foal.jpg");
        pictureBox1.SizeMode = PictureBoxSizeMode.Normal;
        pictureBox1.Dock = DockStyle.Fill;
        this.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;
        this.Controls.Add(label1);
    }
}
```

```

    // Hook up the picturebox to event handlers for doing mousy things

    this.pictureBox1.MouseDown += new MouseEventHandler(MouseButtonIsDown);
    this.pictureBox1.MouseUp += new MouseEventHandler(MouseButtonIsUp);
    this.pictureBox1.MouseMove += new MouseEventHandler(TheMouseMoved);
}

// An event handler that will run when the mouse is MOVED
public void TheMouseMoved(object sender, MouseEventArgs e)
{
    // If the user is holding down a SHIFT key on the keyboard
    if ((Control.ModifierKeys & Keys.Shift) == Keys.Shift)
    {
        // Prepare a drawing surface on the picture
        System.Drawing.Graphics g = this.pictureBox1.CreateGraphics();

        // Use a yellow pen
        System.Drawing.Pen yellowPen = 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();
    }
}

// An event handler that will run when a mouse button is pressed DOWN
public void MouseButtonIsDown(object sender, MouseEventArgs e)
{
    // 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
}

// An event handler that will run when a mouse button is released
public void MouseButtonIsUp(object sender, MouseEventArgs e)
{
    /* 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

    Rectangle r = 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 == MouseButton.Left)
        {
            /* If left button was pressed/released, just draw
               a visual rectangle so they can see it */

            // Prepare a drawing surface on the picture box

            Graphics g = this.pictureBox1.CreateGraphics();

            // Draw a rectangle in red

            Pen redPen = 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);
        }
    }

    // A method that increases the lightness of the image portion
    // selected, by increasing the lightness of each pixel in that area
    public void ChangeLightness(Rectangle rect)
    {
        int newRed, newGreen, newBlue;
        Color pixel;

        // Get a copy of the picture that's in the picture box

        System.Drawing.Bitmap picture = new Bitmap(this.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) || (rect.Height>150) )
        {
            DialogResult result =
                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 == DialogResult.Cancel ) return;
        }
    }

```

```

    }

    /* 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 (int x = rect.X; x < rect.X + rect.Width; x++)
    {
        // And from top to bottom, down the height of the area selected ...

        for (int y = rect.Y; y < (rect.Y + rect.Height); y++)
        {
            // Read the pixel we're at

            pixel = picture.GetPixel(x, y);

            // Prepare a brighter version of the three primary colors in that pixel

            newRed = (int)Math.Round(pixel.R * 2.0, 0);
            if (newRed > 255) newRed = 255;
            newGreen = (int)Math.Round(pixel.G * 2.0, 0);
            if (newGreen > 255) newGreen = 255;
            newBlue = (int)Math.Round(pixel.B * 2.0, 0);
            if (newBlue > 255) newBlue = 255;

            // Apply the new colors to the pixel

            picture.SetPixel
            (
                x,
                y,
                Color.FromArgb((byte)newRed, (byte)newGreen, (byte)newBlue)
            );
        }
    }

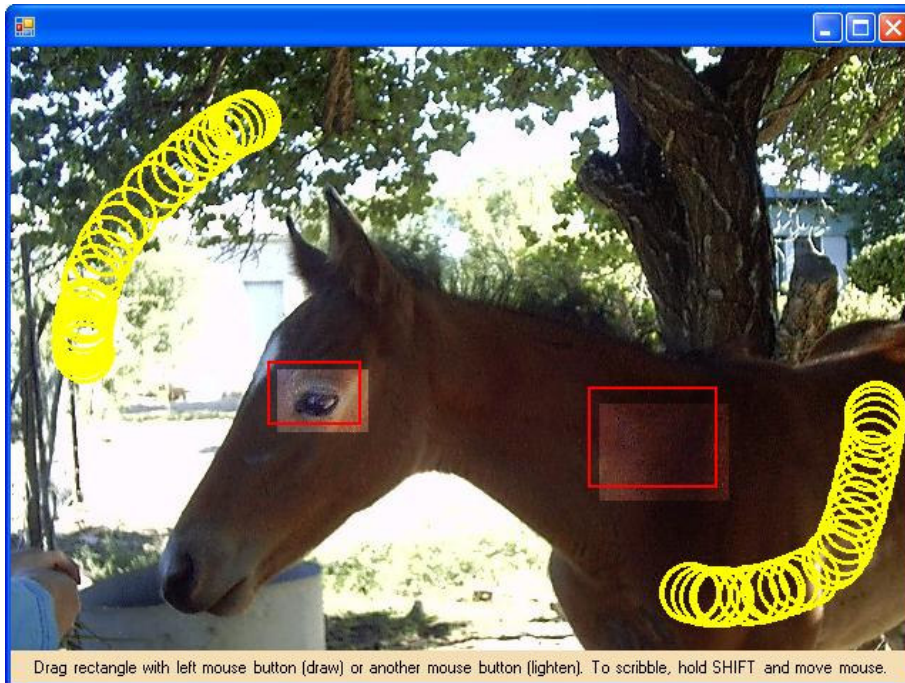
    // Put the modified copy of the picture into the pictureBox
    // so the user can see the changes

    this.pictureBox1.Image = picture;
}

static void Main()
{
    // Create a new instance of our form class

    Application.Run(new FunWithTheMouse());
}
}

```



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 C# 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.

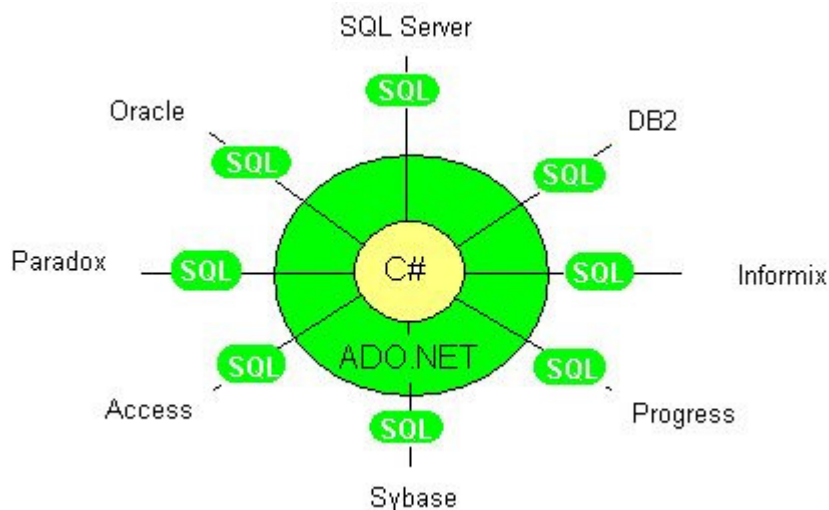
The SQL Language

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 C#? Will they all understand what we're asking for?

The simple answer is that you use a language like C# 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 C#, 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 C# 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 C# 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 C#

In the following C# 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

- ### 3. 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.

۲. 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)

```
using System.Windows.Forms;
using System.Data;
using System.Data.SqlClient;      // Namespace for working with SQL Server data

class SimpleDataAccess : Form
{
    public SimpleDataAccess()
    {
        // Set the window title
        this.Text = "A simple databasing program";

        // Determine the physical path to the PLANETS sample database
        string dbLocation =
            System.IO.Path.GetFullPath("../.../database/SqlServer/planets.mdf");
    }
}
```

```

// Add a label that fills the form

Label label1 = new Label();
label1.Dock = DockStyle.Fill;
this.Controls.Add(label1);

// Connect to the SQL Server database

SqlConnection connection1 = 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

string sql = "SELECT * FROM PLANET";
SqlCommand command1 = new SqlCommand(sql, connection1);
SqlDataReader dataReader1 = command1.ExecuteReader();

// Loop through the records returned and add each planet name to the label

while (dataReader1.Read())
{
    label1.Text = label1.Text + dataReader1["PlanetName"] + "\n";
}

// Clean up

dataReader1.Close();
connection1.Close();
}

static void Main()
{
    // Start a new instance of a forms application, using our class above
    Application.Run(new SimpleDataAccess());
}
}

```



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)

```
using System.Windows.Forms;
using System.Data;
using System.Data.SqlClient;    // Namespace for working with SQL Server databases

class DataInGrid : Form
{
    public DataInGrid()
    {
        // Set the window title
        this.Text = "One-Way Database Grid Binding";

        // Determine the physical path to the PLANETS sample database
        string dbLocation =
            System.IO.Path.GetFullPath("../../../database/SqlServer/planets.mdf");

        // Add a DataGridView to the form

        DataGridView DataGridView1 = new DataGridView();
        DataGridView1.Width = this.Width;
        DataGridView1.Height = 250;
        DataGridView1.DataMember = "Table";
        DataGridView1.Dock = DockStyle.Fill;
        this.Controls.Add(DataGridView1);

        // Connect to the SQL Server database

        SqlConnection connection1 = 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)
        DataSet dataSet1 = new DataSet();

        // The DataAdapter will be the bridge between the database and the dataset
        SqlDataAdapter sqlDataAdapter1 = new SqlDataAdapter();
```

```

// Tell the DataAdapter what we want it to fetch, and where from
sqlDataAdapter1.SelectCommand =
    new SqlCommand("SELECT * FROM PLANET", connection1);

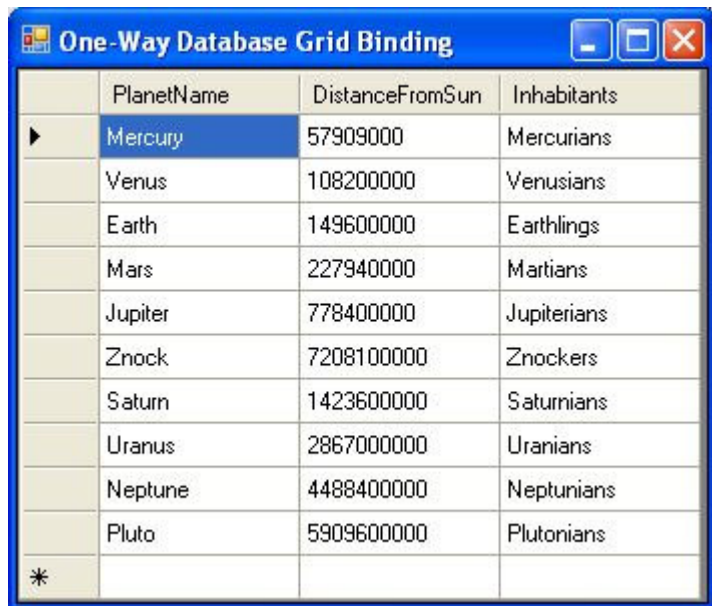
// Fill the in-memory DataSet with the data now
sqlDataAdapter1.Fill(dataSet1);

// Hook the DataGridView (the visual grid) to the in-memory data
DataGridView1.DataSource = dataSet1;

// Close the database connection
connection1.Close();
}

static void Main()
{
    // Start a new instance of a forms application, using our class above
    Application.Run(new DataInGrid());
}
}

```



	PlanetName	DistanceFromSun	Inhabitants
▶	Mercury	57909000	Mercurians
	Venus	108200000	Venusians
	Earth	149600000	Earthlings
	Mars	227940000	Martians
	Jupiter	778400000	Jupiterians
	Znock	7208100000	Znockers
	Saturn	1423600000	Saturnians
	Uranus	2867000000	Uranians
	Neptune	4488400000	Neptunians
	Pluto	5909600000	Plutonians
*			

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)

```
using System.Windows.Forms;
using System.Data;
using System.Data.SqlClient;    // Namespace for working with SQL Server databases

class PlanetsForm : Form
{
    // Declare some objects we'll be talking to from different methods

    DataGridView dg;
    SqlDataAdapter da;

    public PlanetsForm()
    // This is the "constructor" method for the PlanetsForm class
    {
        // Set the window title
        this.Text = "Two-way Database Grid Binding";

        // Determine the physical path to the PLANETS sample database
        string dbLocation =
            System.IO.Path.GetFullPath("../../../database/SqlServer/planets.mdf");

        // Prepare to connect to the SQL Server database
        string connectionString =
            @"data source=.\SQLEXPRESS;" +
            "User Instance=true;Integrated Security=SSPI;AttachDBFilename="
            + dbLocation;

        /* Add a "Save" button to the form */

        Button btnSave = new Button();
        btnSave.Text = "Save";
        btnSave.Click += new System.EventHandler(BtnSave_Click);
        btnSave.Dock = DockStyle.Top;
        this.Controls.Add(btnSave);

        /* Add a DataGridView to the form */

        dg = new DataGridView();
        dg.Width = this.Width;
        dg.Height = 250;
        dg.Dock = DockStyle.Fill;
        this.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
        SqlCommandBuilder cb = new SqlCommandBuilder(da);

        // DataTable will keep track, in memory, of changes
        DataTable dt = new DataTable();
    }
}
```

```

        // Fill the DataTable with the data now
        da.Fill(dt);

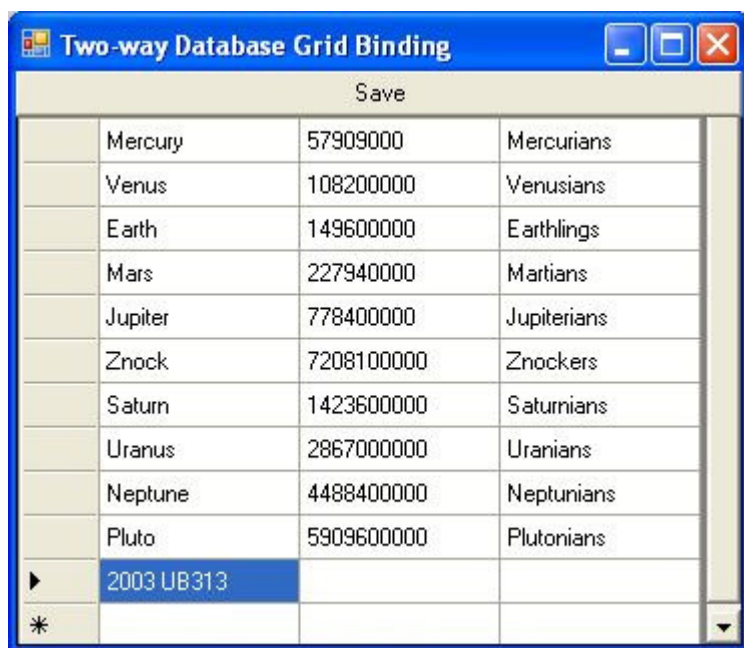
        // Link the DataTable to the DataGridView now
        dg.DataSource = dt;
    }

    static void Main()
    {
        // Start a new instance of a forms application, using our class above
        Application.Run(new PlanetsForm());
    }

    private void BtnSave_Click(object sender, System.EventArgs e)
    {
        // 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((DataTable)dg.DataSource);
        MessageBox.Show
            ("Data has been saved", "For your information", MessageBoxButtons.OK);
    }
}

```



Save			
	Mercury	57909000	Mercurians
	Venus	108200000	Venusians
	Earth	149600000	Earthlings
	Mars	227940000	Martians
	Jupiter	778400000	Jupiterians
	Znock	7208100000	Znockers
	Saturn	1423600000	Saturnians
	Uranus	2867000000	Uranians
	Neptune	4488400000	Neptunians
	Pluto	5909600000	Plutonians
▶	2003 UB313		
*			

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:

```
string connectionString =  
    "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):

```
string connectionString =  
@"data source=.\SQLEXPRESS;Integrated Security=SSPI;  
AttachDBFilename=c:\C#4#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)

A connection string for Microsoft Access could look like this:

```
string connectionString =  
@"Provider=Microsoft.Jet.OLEDB.4.0; Data Source  
c:\C#4#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 C# 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.

```
using 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:

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

</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

```
using System;
using System.Windows.Forms;
using System.Drawing;
using System.Xml;           // Namespace for working with XML generally
using System.Xml.XPath;    // Namespace for working with XML "paths"

class XmlRetriever : Form
{
    ComboBox comboBox1;
    Button button1;
    ListBox listBox1;
    RichTextBox richTextBox1;
    XmlDocument xmlDoc;

    // Constructor method for our class
    public XmlRetriever()
    {
        // Set the window title and height

        this.Text = "XML Retrieval";
        this.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;
    }
}
```

```

        this.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;
        this.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);
        button1.Click += new EventHandler(Button1_Click);
        this.Controls.Add(button1);

        // Add a listbox to display the items returned

        listBox1 = new ListBox();
        listBox1.Dock = DockStyle.Bottom;
        listBox1.Location = new Point(10, 10);
        this.Controls.Add(listBox1);
    }

    static void Main()
    {
        // Start a new instance of a forms application, using our class above
        Application.Run( new XmlRetriever() );
    }

    // An eventhandler that will run when the button is clicked
    void Button1_Click(object sender, EventArgs e)
    {
        XmlNodeList xmlNodes;
        XmlNode xmlElement;
        string elementValue;

        // 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();

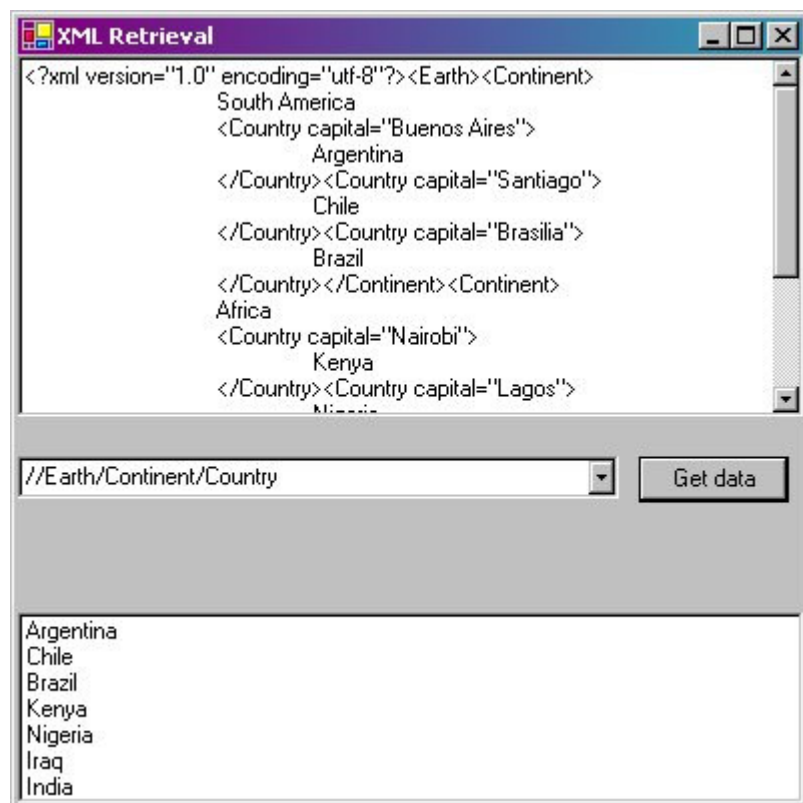
```

```

        // Loop through all the matching items that were found,
        // adding each item to the listbox
        for (int i = 0; i < xmlNodes.Count; i++)
        {
            xmlElement = xmlNodes[i];
            if (xmlElement.HasChildNodes)
            {
                elementValue = xmlElement.FirstChild.Value.Trim();
                listBox1.Items.Add(elementValue);
            }
        }
    }
}
catch (XPathException ex)
{
    // 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
    );
}
}
}

```

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

Microsoft's newest programming language, C#, (pronounced "c-sharp") is both powerful and easy to use. 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.