1 Car journey calculation

This book is about creating web page applications. However, we will begin by producing a C# application to run on a stand-alone Microsoft Windows computer in order to focus on the principles of programming. We will then produce exactly the same application as a web page version, to demonstrate the similarities and differences between the programming environments for web pages and windows forms when using Microsoft Visual Studio.

We will carry out calculations for planning a car journey. The program will input:

- The journey distance, entered either in miles or kilometres.
- The fuel usage of the vehicle, in miles per gallon.
- The price of fuel per litre.

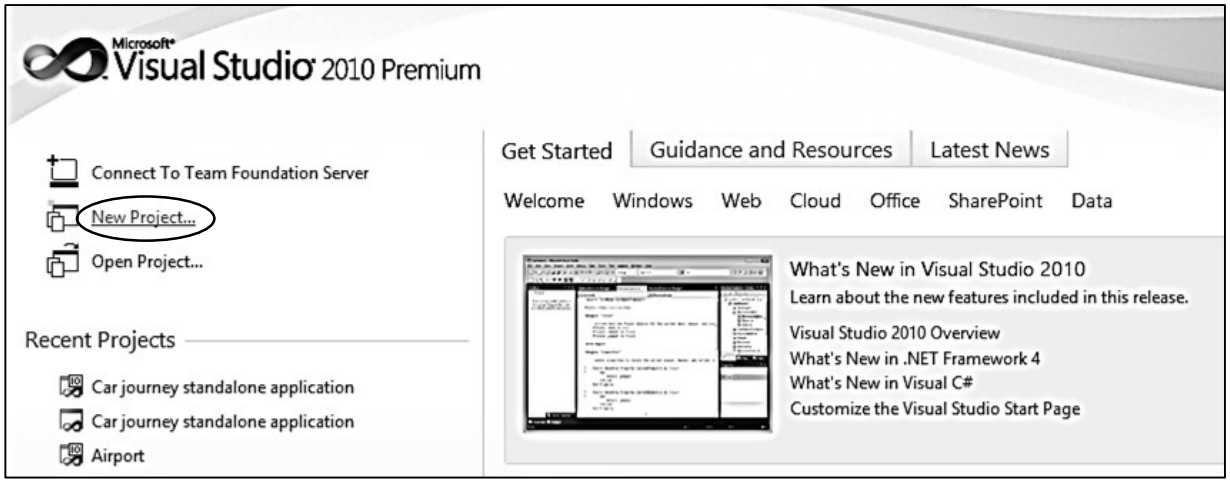
The program will calculate the quantity of fuel required for the journey and the total fuel cost.

We will then make an estimate of the journey time in hours and minutes. To do this, the program will also input:

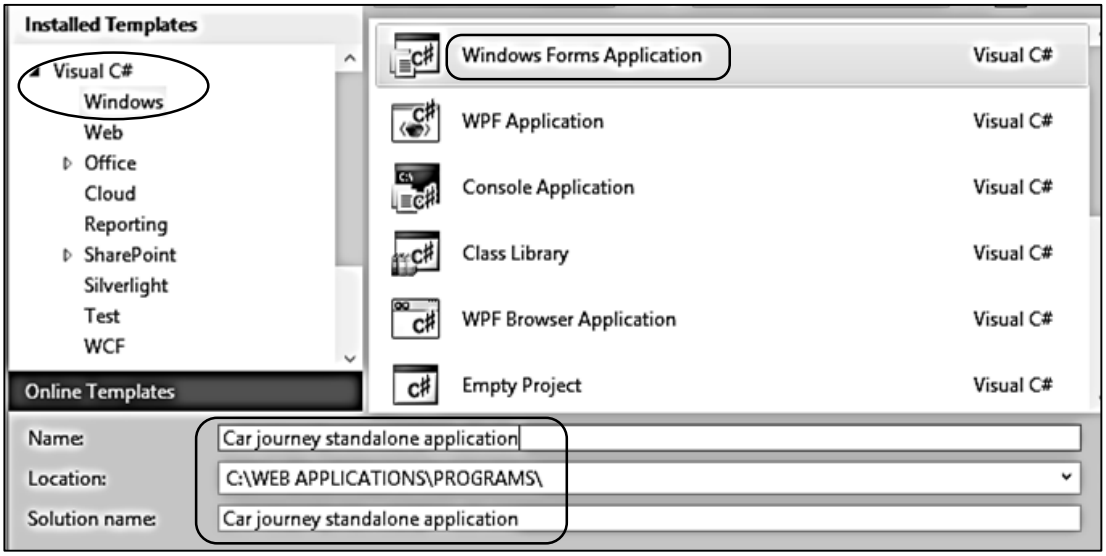
- The percentage of the total journey by motorway.
- The number of town centres which have to be crossed on route.

We will base the calculation on an average speed of 60mph on motorways and 35mph on other roads. We will add 20 minutes for each town centre crossed.

Begin by opening Microsoft Visual Studio 2010 and select ***New Project***:

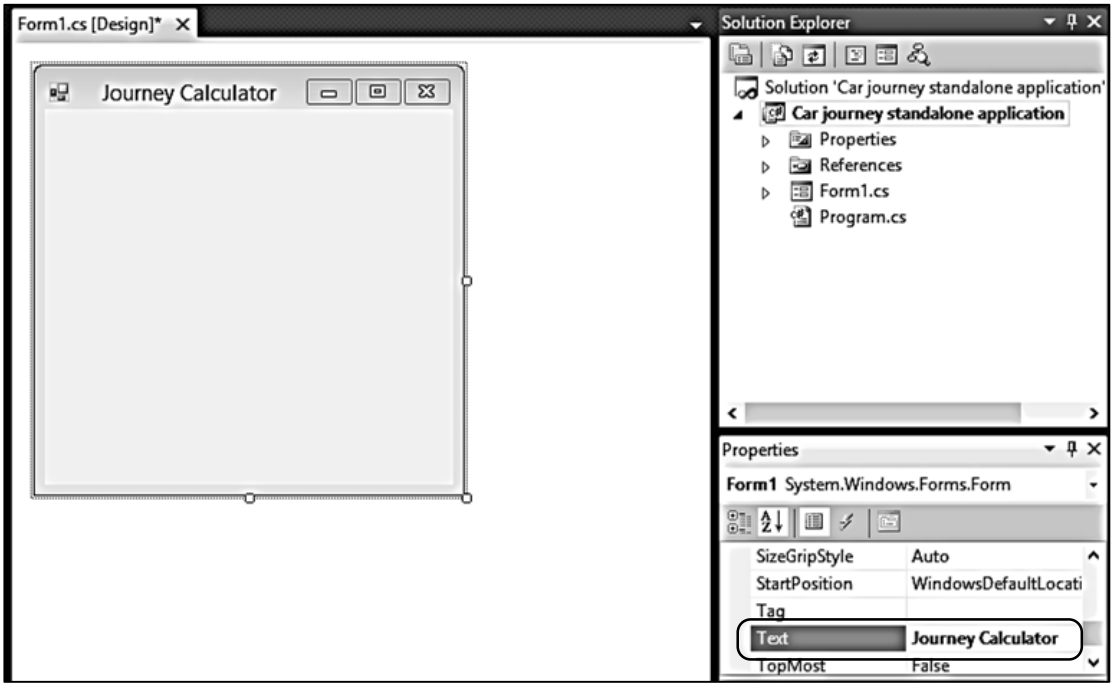


Select ***Visual C# Windows***, then ***Windows Forms Application***. Give the name ***'Car journey standalone application'*** and select a folder location to store the project.



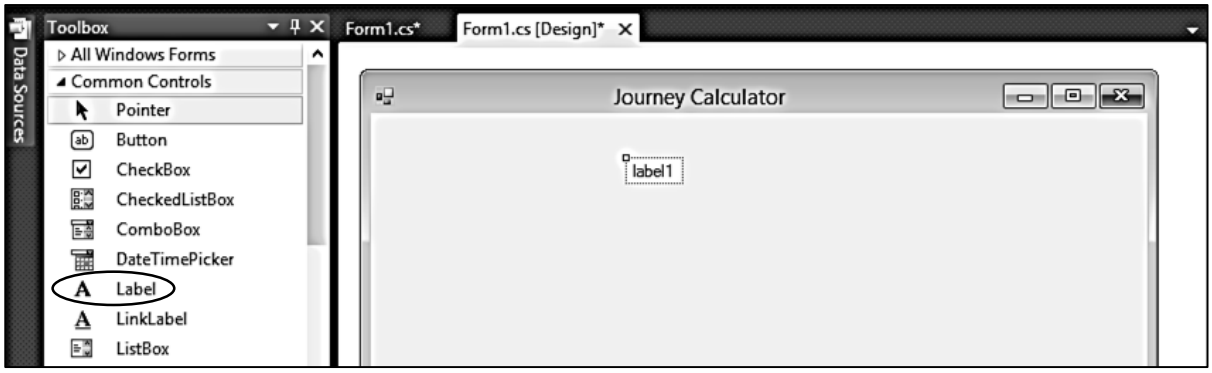
Click the ***OK*** button to continue. A blank form will appear. Go to the ***Properties*** window in the bottom right of the screen and change the ***Text*** property to ***'Journey Calculator'***. This text should be displayed on the window header bar.

Note: if the ***Properties*** window is not visible, then right click the form and select the ***Properties*** option from the pop-up menu.



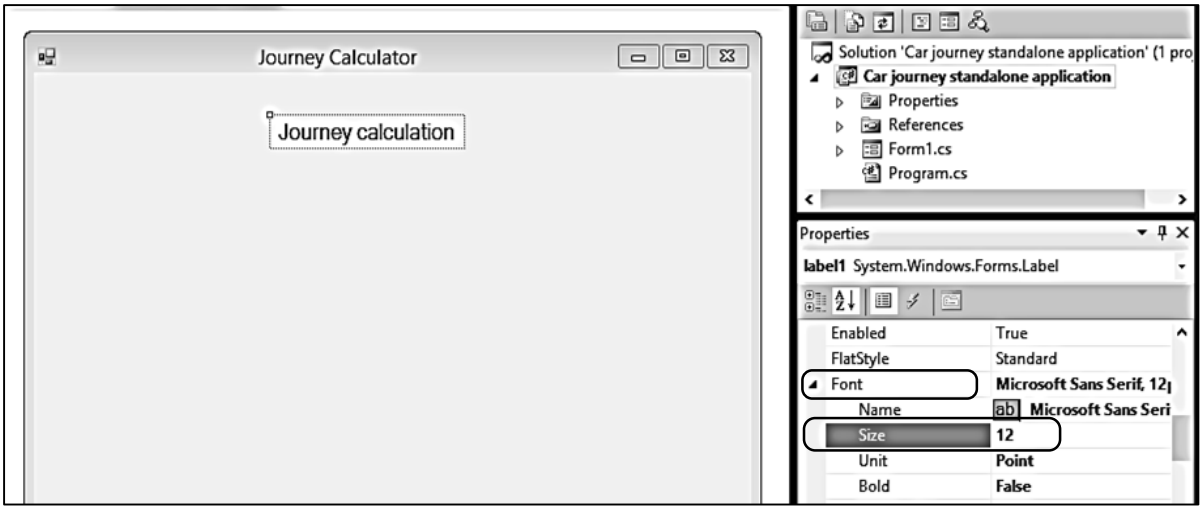
Select the Label component from the Toolbox window. Drag and drop a label on the form.

If the Toolbox window is not visible, open this by going to the ***View*** option on the menu bar at the top of the screen and selecting ***Toolbox*** from the drop-down list.



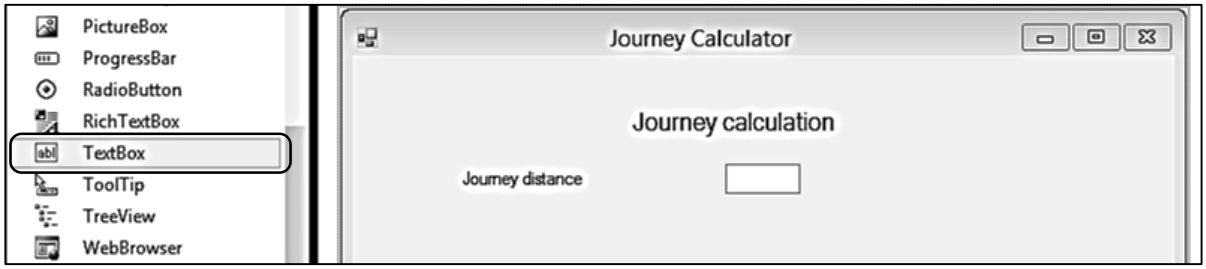
Go to the ***Properties*** window and set the ***Text*** property to '***Journey calculation***'.

Click the small triangle icon alongside the ***Font*** property to display further options. Set ***Size*** to 12point.

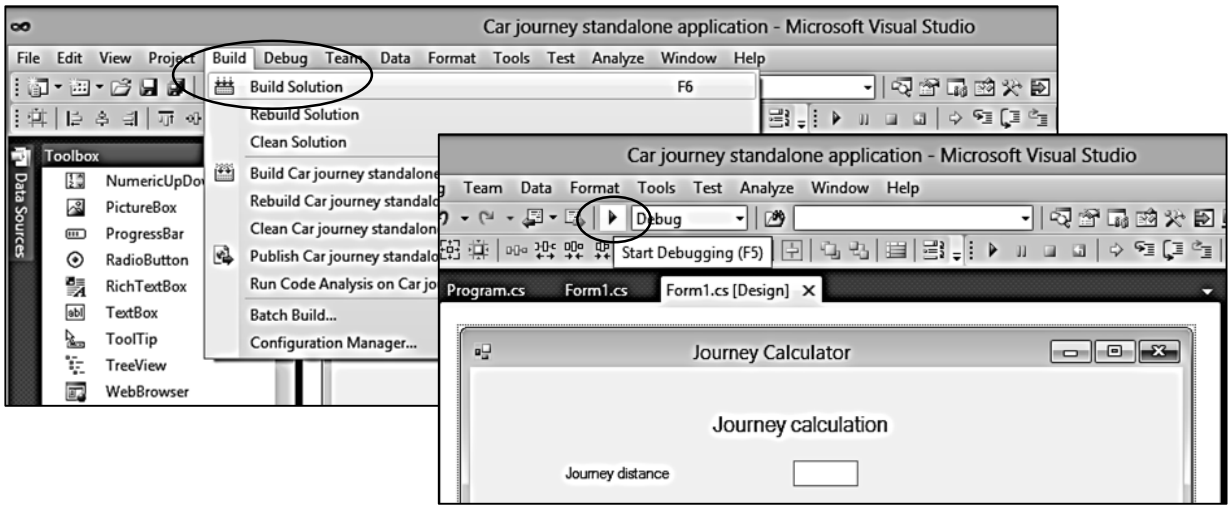


Add another label and set the ***Text*** property to '***Journey distance***'.

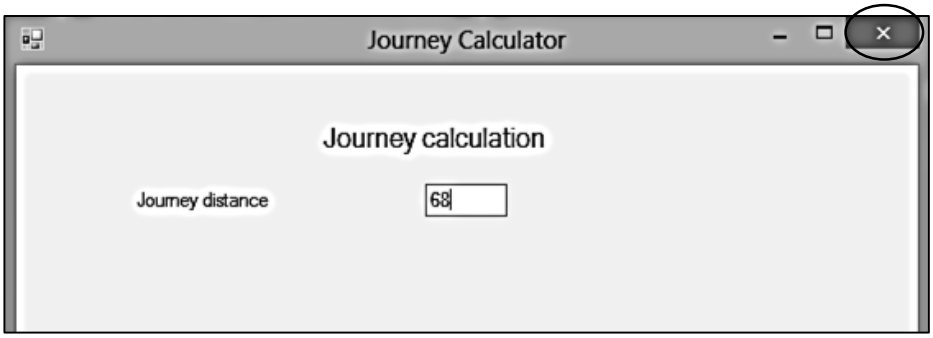
We will now add a ***TextBox*** component which will allow the user to input a value when the program is run. Select ***TextBox*** from the ***Toolbox*** window and drag this onto the form. The input will only be a two or three digit number, so we can reduce the length of the TextBox. To do this, select the box and drag the small square at the edge of the component.



At this stage it would be good to test the program which we have developed so far. Go to the menu bar at the top of the screen and select ***Build*** / ***Build Solution***. If no error messages appear, then all is well. Click the ***green arrow*** icon below the menu bar to run the program.

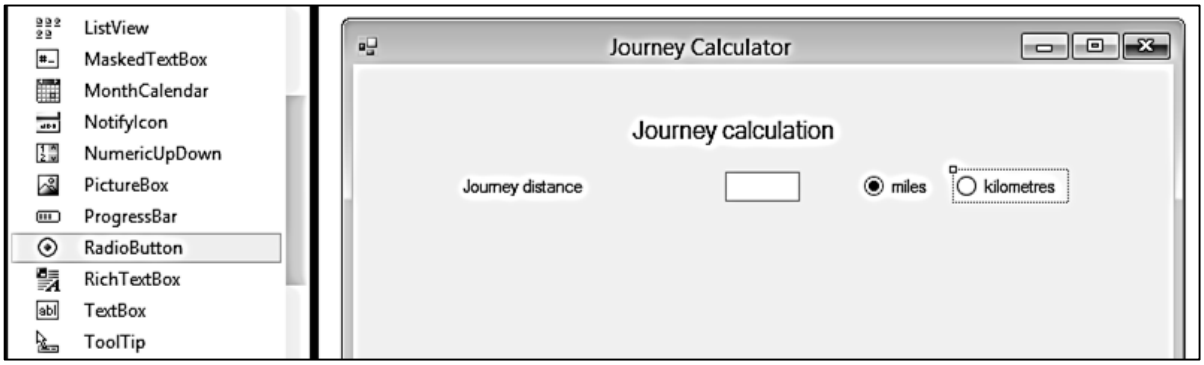


The program should run, displaying the form that we have created. It will be possible to type data into the TextBox.

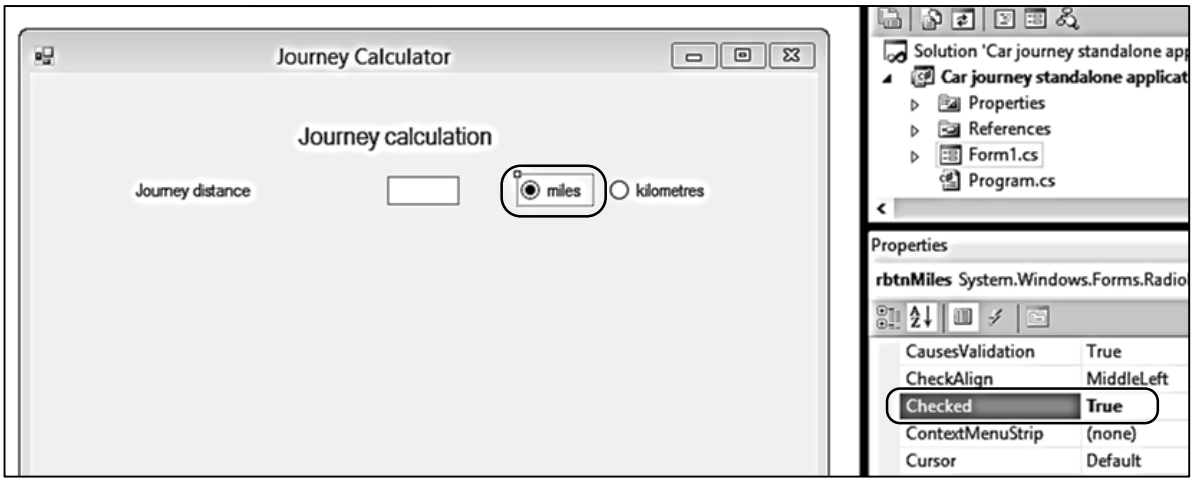


Click the red cross in the corner of the form to stop the program and return to the editing screen.

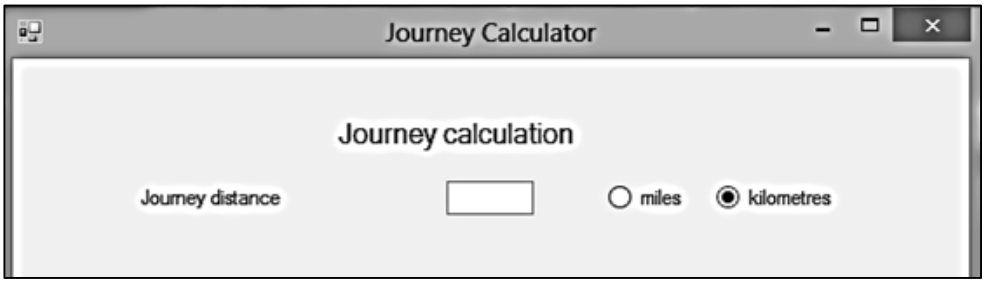
We will allow the user to enter the journey distance as either miles or kilometres. To do this, drag the edge of the form to make it wider, then add two ***RadioButton*** components from the ***Toolbox***. To create the captions '***miles***' and '***kilometres***', select each RadioButton and change its ***Text*** property.



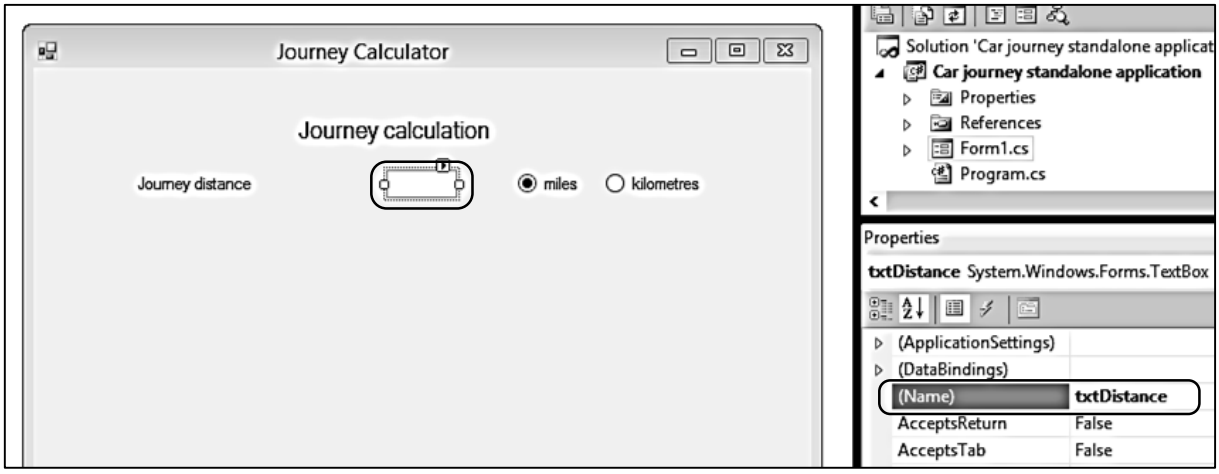
An important feature of a radio button group is that only one button can be selected at a time. We will set '***miles***' to be the default selection which appears when the program is first run. To do this, select the '***miles***' RadioButton and set the ***Checked*** property to '***True***'.



Run the program and check that the radio buttons repond correctly.

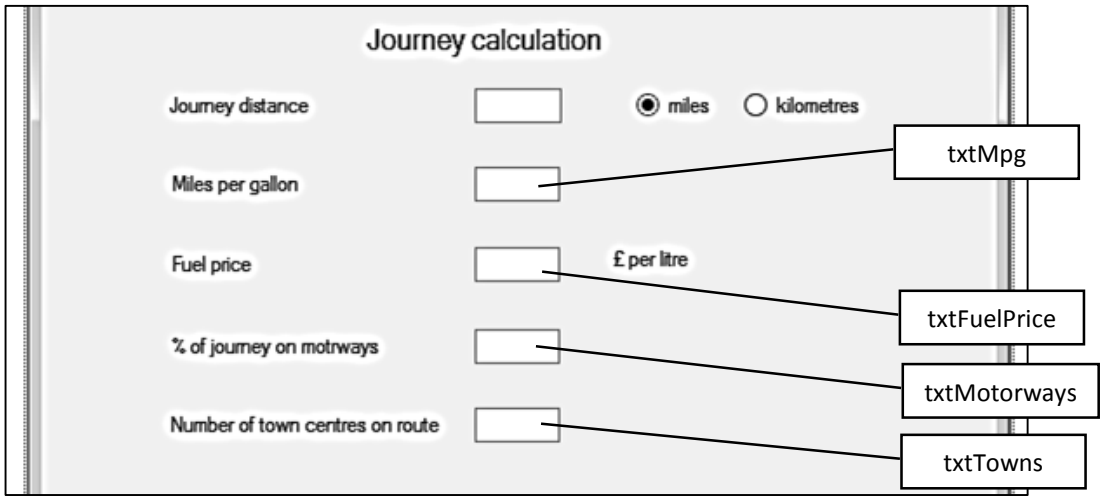


Return to the editing screen. Before continuing with the program, it will be good to give descriptive names to the input components we have created. Select the ***TextBox*** and rename this as '***txtDistance***'. The use of descriptive names for components will make the programming easier to understand when we come to the calculation stage.

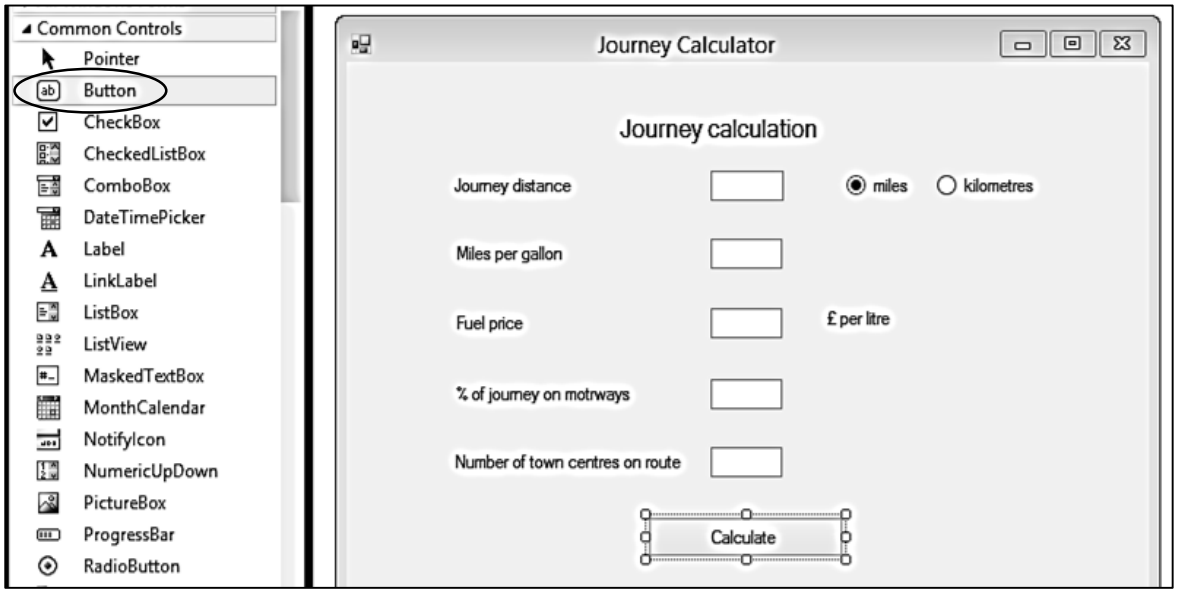


In a similar way, give the names '***rbtnMiles***' and '***rbtnKm***' to the two radio buttons. To do this, select each component, go to the ***Properties*** window, and change the ***Name*** property.

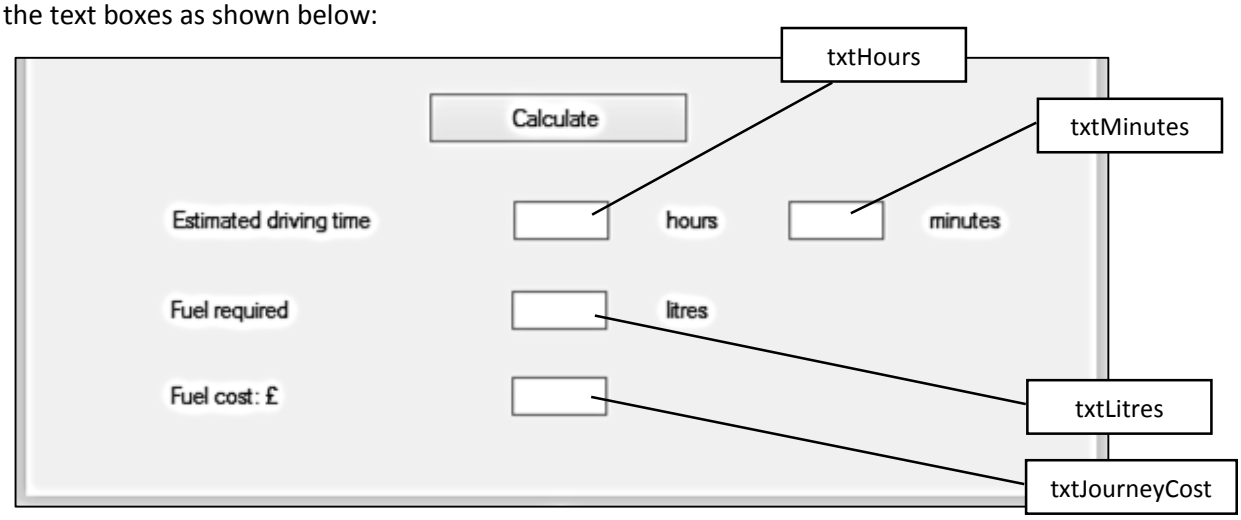
We will continue to construct the input form by adding labels and text boxes for the other data required by the journey calculations. Give names to the text boxes as shown:



Add a ***Button*** component from the Toolbox. Set the ***Name*** property of the component to '***btnCalculate***'. Change the caption on the button by setting the ***Text*** property to '***Calculate***'.



Complete the form by adding labels and text boxes for output of the results. Drag the bottom edge of the form downwards if necessary, to make space for the additional components. Give names to the text boxes as shown below:



We can now work on the calculation process. Double click the '***Calculate***' button on the form, and a C# code window will open. The computer has created a ***button_click*** method for you. Program code which we put into this method will operate when the user clicks the button while the program is running.

```
using System.Text;
using System.Windows.Forms;

namespace Car_journey_standalone_application
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {

        }

        private void btnCalculate_Click(object sender, EventArgs e)
        {

        }
    }
}
```

Add the following lines of code to the button_click method:

```
private void btnCalculate_Click(object sender, EventArgs e)
{
    try
    {
        //get journey distance in miles
        Double distance = Convert.ToDouble(txtDistance.Text);
    }
    catch
    {
        MessageBox.Show("Incorrect data entered");
    }
}
```

The line:

```
//get journey distance in miles
```

is simply a comment to help any future programmer to understand the purpose of this section of the code. The two ***forward-slash*** symbols instruct the computer to ignore anything written on this line when the program runs.

The line:

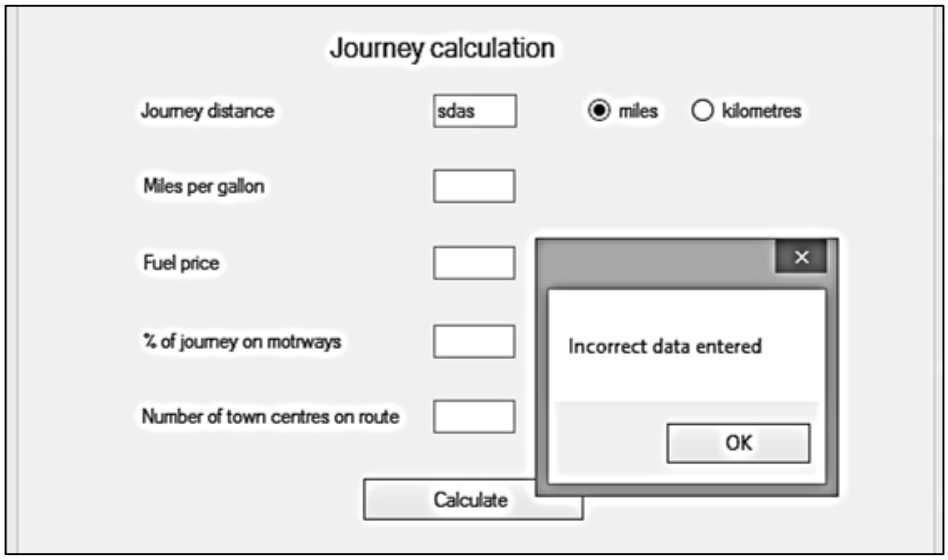
```
Double distance = Convert.ToDouble(txtDistance.Text);
```

instructs the program to take whatever is written in the ***txtDistance*** text box and convert it into a decimal number. This will then be stored as a variable called ***distance***, for use in the calcuation.

We anticipate that things may go wrong when the program runs! For example, the user might accidentally enter letters of the alphabet into the text box instead of numbers. The program would then be unable to create a valid decimal number, leading to a program crash. To avoid this, we use a ***try…catch*** structure. If the text box input cannot be converted to a decimal number, then a message box will appear:

```
MessageBox.Show("Incorrect data entered");
```

Build and run the program. Enter a correct number in the Distance text box and click the Calculate button. At this stage, nothing will happen. Now repeat the process, but put letters into the Distance text box. The error message will appear, and the user will have an opportunity to correct their error.



We would like the ***distance*** variable to represent the journey distance in miles. This will already be the case if the user selected the ***miles*** radio button. However, we will need to carry out a conversion calculation if the ***kilometres*** radio button was selected.

1 kilometre = 0.621 miles

Add code to identify the ***kilometres*** radio button being selected, and carry out the conversion to miles.

```
try
 {
     //get journey distance in miles
     Double distance = Convert.ToDouble(txtDistance.Text);
     if (rbtnKm.Checked == true)
     {
         distance = distance * 0.621;
     }
 }
```

We can now collect the data for fuel consumption, and use this to calculate the number of gallons of fuel required for the journey. We make use of the distance ***variable*** obtained earlier and create two new variables, ***mpg*** and ***fuelRequired***.

```
try
{
    //get journey distance in miles
    Double distance = Convert.ToDouble(txtDistance.Text);
    if (rbtnKm.Checked == true)
    {
        distance = distance * 0.621;
    }

    //calculate fuel required
    Double mpg = Convert.ToDouble(txtMpg.Text);

    //fuel required in gallons
    Double fuelRequired = distance / mpg;
}
```

The program should output the quantity of fuel required in litres, since this is the normal unit of measurement at petrol filling stations. A conversion is required:

1 gallon = 4.546 litres

Once the number of litres has been found, the result can be displayed in the ***txtLitres*** text box. However, we should be careful that we output data to a sensible number of decimal places. In this case, one decimal place will be sufficient. Add lines of code to the calculation method:

```
    //calculate fuel required
    Double mpg = Convert.ToDouble(txtMpg.Text);

    //fuel required in gallons
    Double fuelRequired = distance / mpg;

    //convert to litres
    fuelRequired = fuelRequired * 4.546;

    //output litres of fuel (1 decimal place)
    string output = fuelRequired.ToString("f1");
    txtLitres.Text = output;
}
```

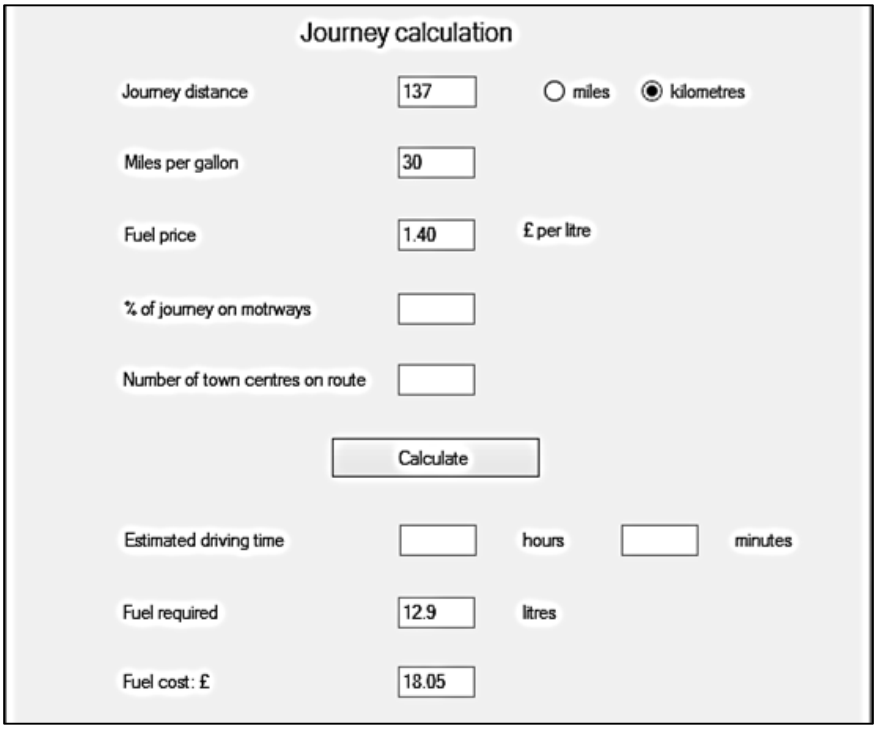
Run the program and enter values for journey distance and fuel consumption (miles per gallon). Click the Calculate button, and check that the quantity of fuel for the journey (litres) is shown.

It is sensible to independently check the results of all program calculations, using a calculator or spreadsheet, to ensure that no mistakes have been made in the program formulae. You might find that there is one penny difference in the result, due to different numbers of decimal places being used by the claculator and the computer program during the calculations.

Return to the program. Add lines of code to input the cost of fuel, calculate the total cost for the journey, then output the result. In this case, we will display the result to two decimal places to represent pounds and pence.

```
    //fuel required in gallons
    Double fuelRequired = distance / mpg;

    //convert to litres
    fuelRequired = fuelRequired * 4.546;

    //output litres of fuel (1 decimal place)
    string output = fuelRequired.ToString("f1");
    txtLitres.Text = output;

    //calculate cost of fuel
    Double fuelPrice = Convert.ToDouble(txtFuelPrice.Text);
    Double totalCost = fuelRequired * fuelPrice;

    //output fuel cost (2 decimal places)
    output = totalCost.ToString("f2");
    txtJourneyCost.Text = Convert.ToString(output);
}
```

Run the program. Enter data for journey distance (either as miles or kilometres), fuel consumption in miles per gallon, and fuel price. Click the Calculate button and check that correct results are given for the fuel required and the total cost of the journey.



The last stage in the calculation is to estimate the journey time. To do this, we will need to collect the data for the percentage of the journey by motorway and the number of town centres crossed on route. The number of town centres must be entered as a whole number, so we will convert this to ***integer*** format.

```
    //output fuel cost (2 decimal places)
    output = totalCost.ToString("f2");
    txtJourneyCost.Text = Convert.ToString(output);

    //get % motorway travel and number of town centres crossed
    Double motorwayPercent = Convert.ToDouble(txtMotorways.Text);
    int townCentres = Convert.ToInt16(txtTowns.Text);
}
```

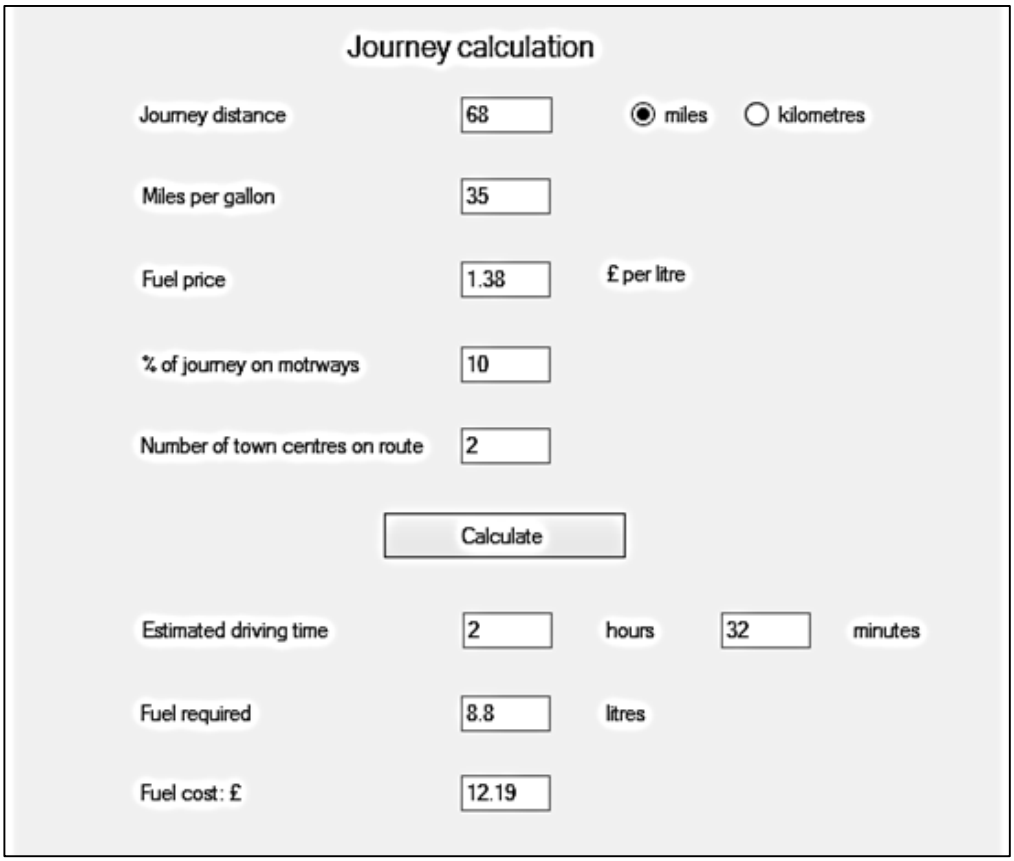
The calculation of journey time involves several stages:

- We use the input percentage of motorway travel and total journey distance to find the miles which will be travelled by motorways and miles travelled on other roads.
- We then calculate journey times in minutes for travelling these distances, assuming an average speed of 60mph on motorways and 35mph on other roads.
- Twenty minutes are added to the journey time for each town centre which has to be crossed on route.
- The total journey time in minutes has now been found. We convert this to separate integer values for hours and minutes.

The '**/ 60**' operator is used to find the number of times that 60 divides into the minute total, ignoring any remainder. The '**% 60**' operator is used to find the remainder when 60 is subtracted from the minutes total as many times as possible.

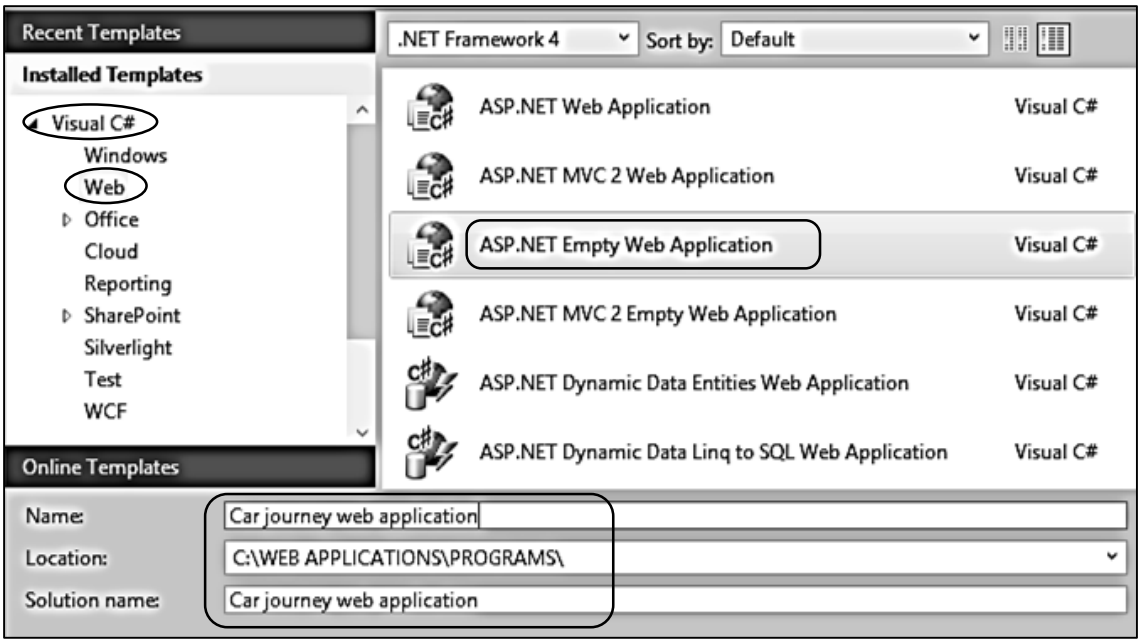
```
    //get % motorway travel and number of town centres crossed
    Double motorwayPercent = Convert.ToDouble(txtMotorways.Text);
    int townCentres = Convert.ToInt16(txtTowns.Text);

    //calculate number of miles travelled on motorways and other roads
    Double motorwayMiles = distance * motorwayPercent / 100;
    Double otherMiles = distance - motorwayMiles;

    //calculate road journey in hours, then multiply by 60 to convert to minutes
    Double totalMinutes = (motorwayMiles / 60 + otherMiles / 35) * 60;

    //add 20 minutes for each town centre crossed
    totalMinutes = totalMinutes + (townCentres * 20);

    //find the journey time in hours and minutes
    int hours = Convert.ToInt16(totalMinutes) / 60;
    int minutes = Convert.ToInt16(totalMinutes) % 60;

    //output the results
    txtHours.Text = Convert.ToString(hours);
    txtMinutes.Text = Convert.ToString(minutes);
}
```

Run the program and check that journey time is calculated correctly.



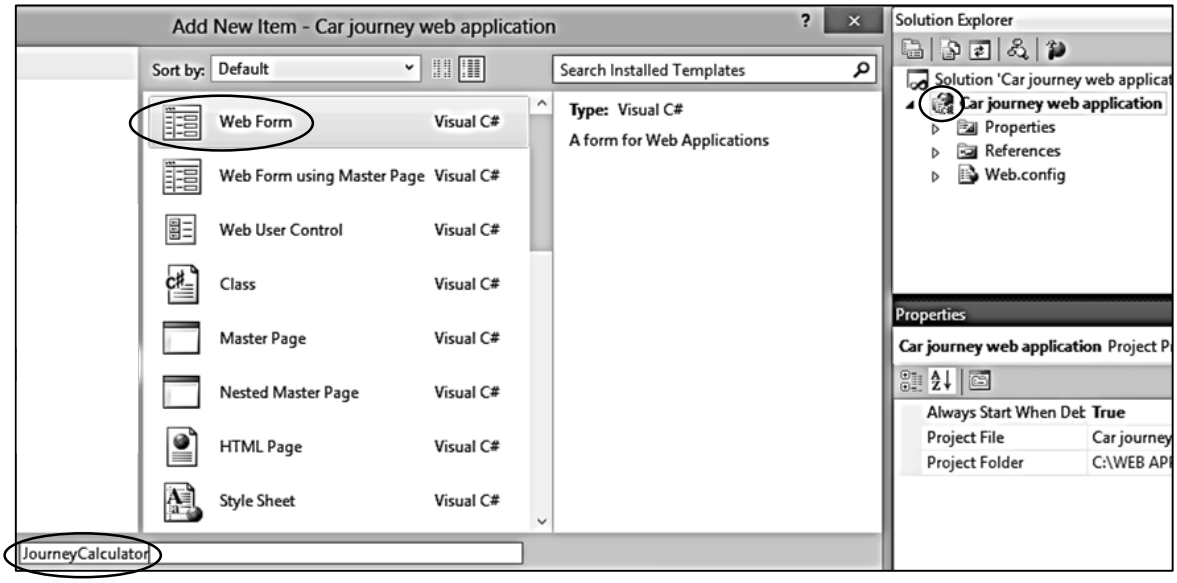
This completes the stand-alone version of the Journey Calculation program. We will now create the same application to run on an Internet page. Fortunately you will discover that the C# code for the calculation is identical, and can be copied directly from the stand-alone version. It is only the web page user interface that will require different program code.

Save the stand-alone program and close Visual Studio. Reopen Visual Studio and select ***New Project***. Select ***Visual C# Web*** and click on ***ASP.NET Empty Web Application***. Give a name for the project, and specify where it should be stored on your computer.



Go to the ***Solution Explorer*** window and right click the ***Car Journey web application*** project icon.

Select ***Add /New Item***, then click ***Web Form***, and give the name ***JourneyCalculator***.



An HTML code page will open. This begins with several lines which specify the programming environment to be used.

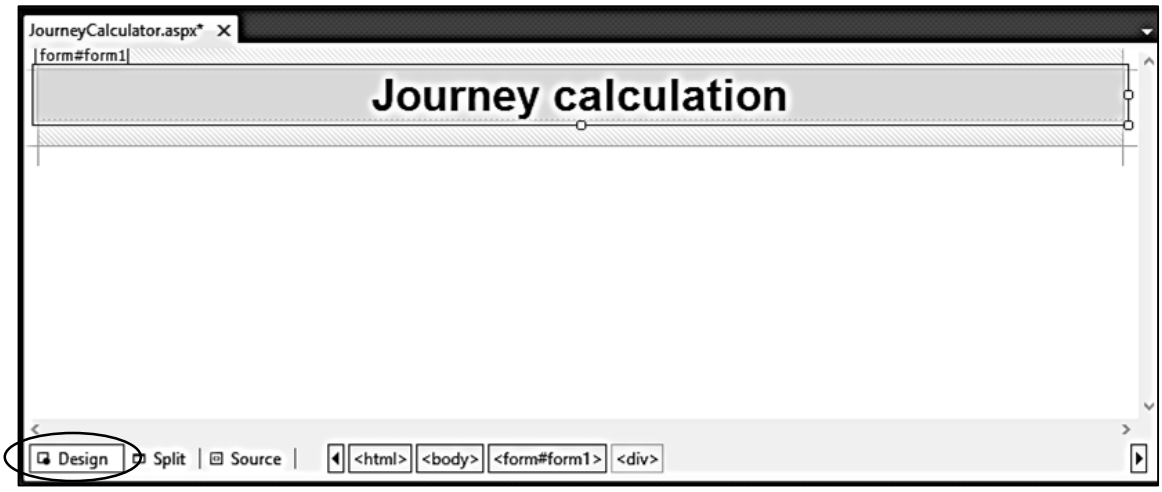
The first line which we need to alter is marked by the ***<title>*** tag. Insert '***Journey Calculator***' as the title which will appear on the web page header bar in the browser.

```
<%@ Page Language="C#" AutoEventWireup="true"
                                  CodeBehind="JourneyCalculator.aspx.cs"
Inherits="Car_journey_web_application.JourneyCalculator" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Journey Calculator</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    </div>
    </form>
</body>
</html>
```

We will now add a title '***Journey calculation***' which will appear on the actual webpage. A title is specified with a heading tag ***<h1>***. We will choose for the title to be centred on the page. Code has also been added to the division tag *** *** to select ***Arial*** font.

```
<head runat="server">
    <title>Journey Calculator</title>
</head>
<body>
    <form id="form1" runat="server">

    <div style="font-family:Arial,Helvetica,sans-serif; font-size:small;">
            <h1 align="center">Journey calculation</h1>

    </div>
    </form>
</body>
</html>
```

The design of the page so far can be checked by using the Design button at the bottom left of the code window.

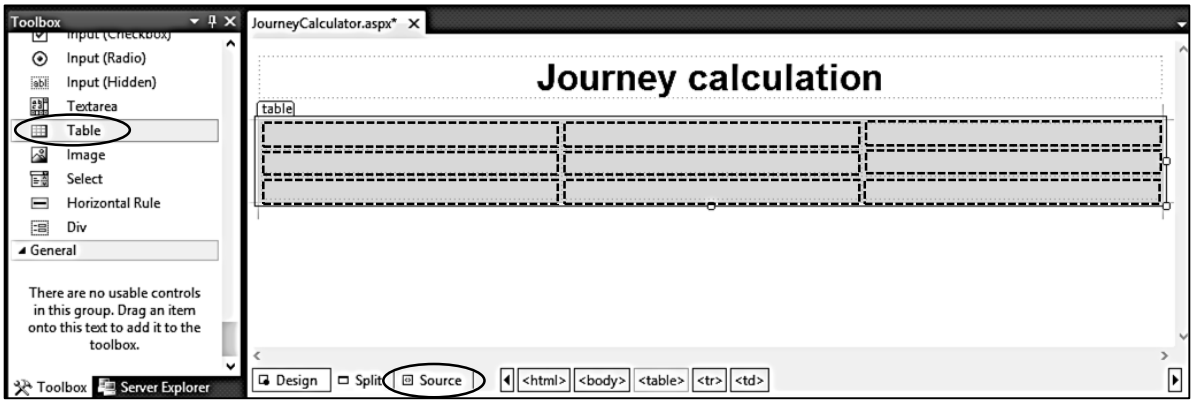


We will add the same set of labels, text boxes and radio buttons to the page as we did earlier for the stand-alone version of the program. However, it is not possible to simply drag and drop components in the correct positions when creating a web page. Instead we must write HTML code to specify the component positions.

A simple way to lay out components on a form is to use an HTML table consisting of five rows of three columns:

Journey distance		◉ miles ○ kilometres
Miles per gallon		
Fuel price		£ per litre
% of journey on motorways		
Number of town centres on route		

A basic table structure can be easily inserted from the design screen. Scroll down the Toolbox list to the HTML section. Select Table and drag this onto the page below the heading. A faint dotted outine of a table will appear, with three rows of three columns.



Click the Source button to return to the HTML code page. Notice that code has been added to the program. The new code is enclosed by the ***<table>*** and ***</table>*** tags. The table code must be inside the ***<form>*** and *** *** tags. If this is not the case then highlight and drag the ***<form>*** and *** *** tags to the correct position:

```
<form id="form1" runat="server">
<div style="font-family: Arial, Helvetica, sans-serif; font-size: small;">
        <h1 align="center">Journey calculation</h1>

<table style="width:100%;">
    <tr>
        <td>
             </td>
        <td>
             </td>
        <td>
             </td>
    </tr>
    <tr>
        <td>
             </td>
        <td>
             </td>
        <td>
             </td>
    </tr>
    <tr>
        <td>
             </td>
        <td>
             </td>
        <td>
             </td>
    </tr>
</table>

</div>
</form>
```

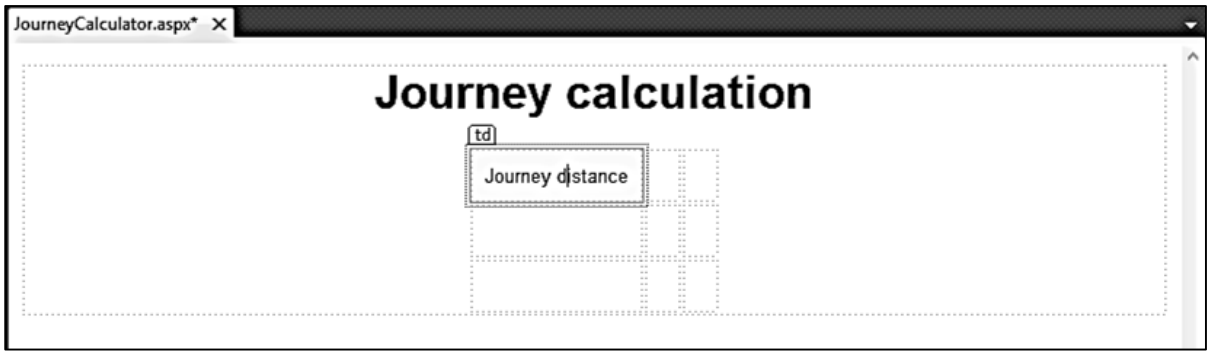
Row 1
Column 1
Column 2
Column 3
Row 2
Column 1
Column 2
Column 3
Row 3
Column 1
Column 2
Column 3

Notice that each new line of the table begins with a ***table row <tr>*** tag. Each new column within a row begins with a ***table data <td>*** tag. The program has put space characters ( ) within each cell of the table, which we can replace with our own labels, text boxes or radio buttons.

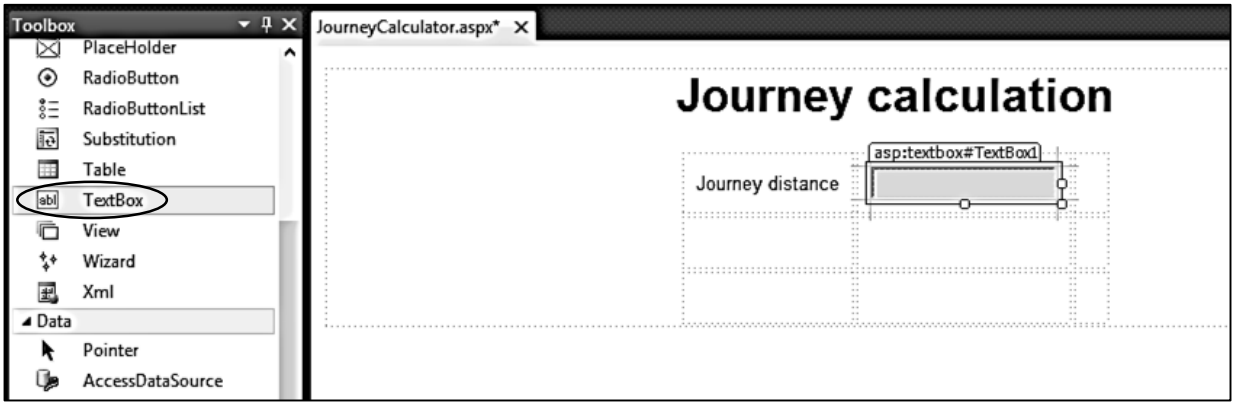
We will begin work on the first row of the table. We will firstly modify the ***<table>*** tag to centre the table on the page, hide the border lines between the table cells, and provide a 10 pixel border space within each cell. We can then insert the words '***Journey distance***' into the first cell.

```
<div style="font-family: Arial, Helvetica, sans-serif; font-size: small;">
    <h1 align="center">Journey calculation</h1>

    <table align="center" border="0" cellpadding=10>
        <tr>
            <td>
                Journey distance
            </td>
            <td>
                 </td>
            <td>
                 </td>
        </tr>
```

Click the Design button at the bottom left of the code window to view the page layout so far. The caption 'Journey distance' should be occupying the left hand cell on the top row of the table.



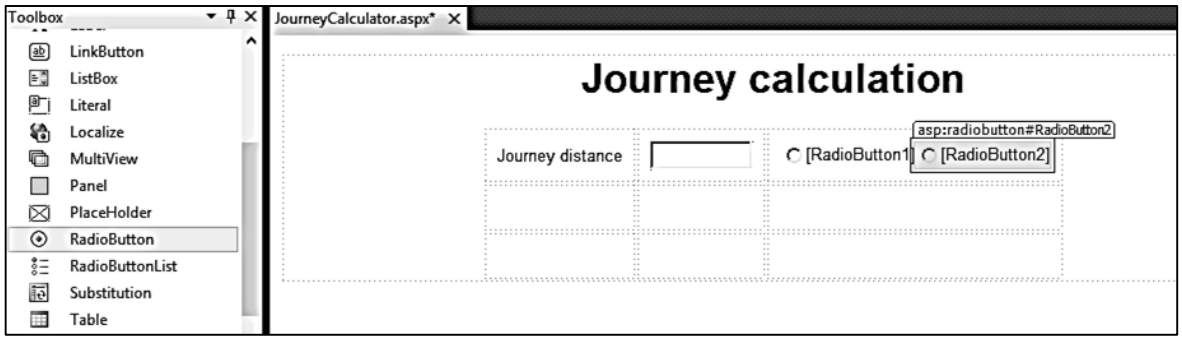
Scroll back to the top of the ***Toolbox*** list and locate the ***TextBox*** component. Drag and drop this into the second cell on the top row of the table.



Return to the HTML code page by clicking the ***Source*** button at the bottom of the window. Notice that the computer has inserted a ***<TextBox>*** tag. We will give this text box the name '***txtDistance***', as we did in the stand-alone program. Also set the width of the text box to 80 pixels:

```
<table align="center" border="0" cellpadding=10>
<tr>
    <td>
        Journey distance
    </td>
    <td>

        <asp:TextBox ID="txtDistance" runat="server" Width="80px">
        </asp:TextBox>

    </td>
    <td>
         </td>
</tr>
```

The final requirement for the top row of the table is the pair of radio buttons to input a choice of miles or kilomtres. Return to the **Design** window and select the ***RadioButton*** component in the ***Toolbox***. Drag two radio buttons to the right hand cell of the top grid row.

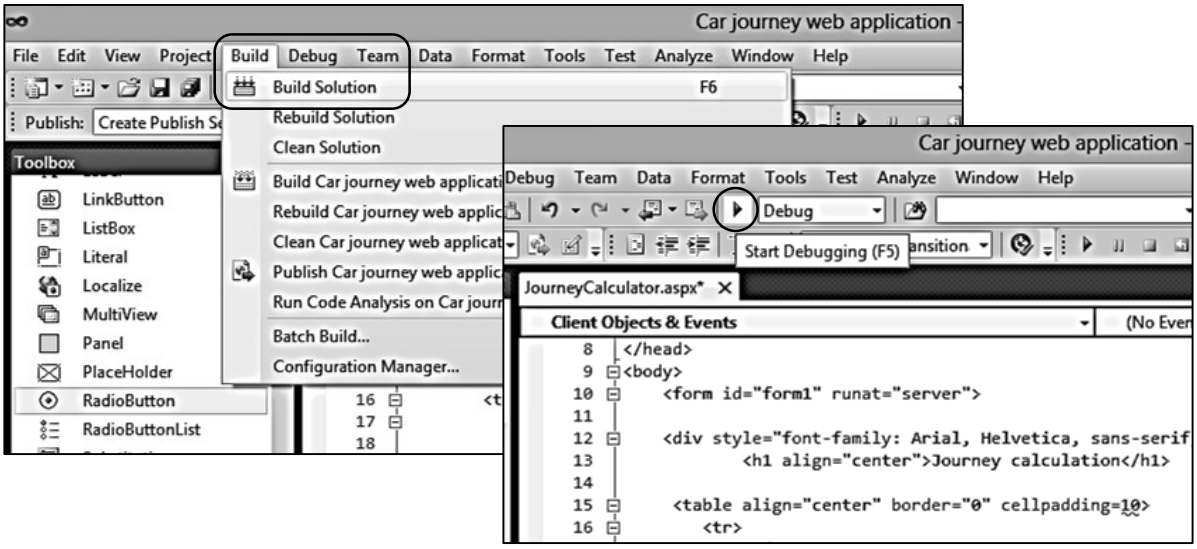


Change to the HTML code page and identify the two ***<RadioButton>*** tags which have been inserted. We need to make some changes to the code, as shown below.

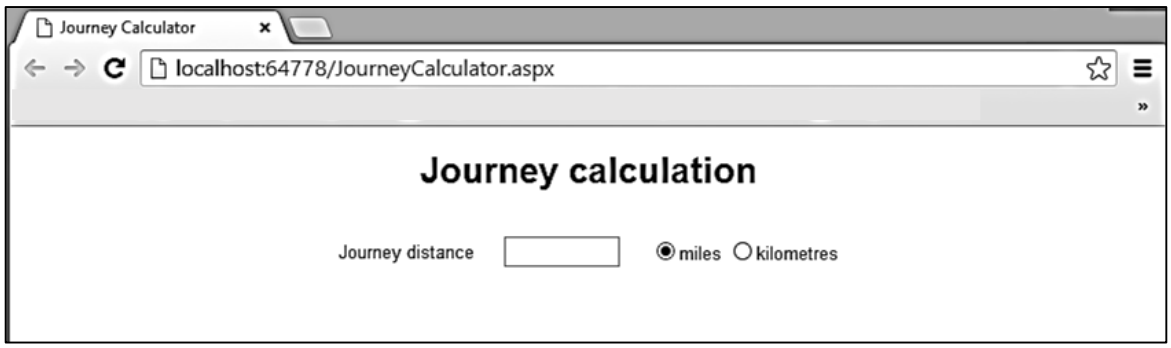
```
<td>
    Journey distance
</td>
<td>
    <asp:TextBox ID="txtDistance" runat="server" Width="80px">
    </asp:TextBox>
</td>
<td>

    <asp:RadioButton ID="rbtnMiles" runat="server" Checked="True"
                    GroupName="distanceUnit" Text="miles" />
    <asp:RadioButton ID="rbtnKm" runat="server" GroupName="distanceUnit"
                    Text="kilometres" />

 </td>
```

We have given the names '***rbtnMiles***' and '***rbtnKm***' to the radio buttons, as in the stand-alone program. We have told the computer to treat both radio buttons as belonging to a group which we have called '***distanceUnit***'. This will ensure that only one of the buttons can be selected. The ***Checked*** property of the '***miles***' radio button is set to '***True***', so that this button is shown as selected when the program first runs.

This is a good moment to test the web page in an internet browser. To do this, go to the menu bar and click Build / Build Solution. If all is well, click the green arrow icon below the menu bar.



The web page should be displayed in your default web browser, as it would appear if published on the Internet.

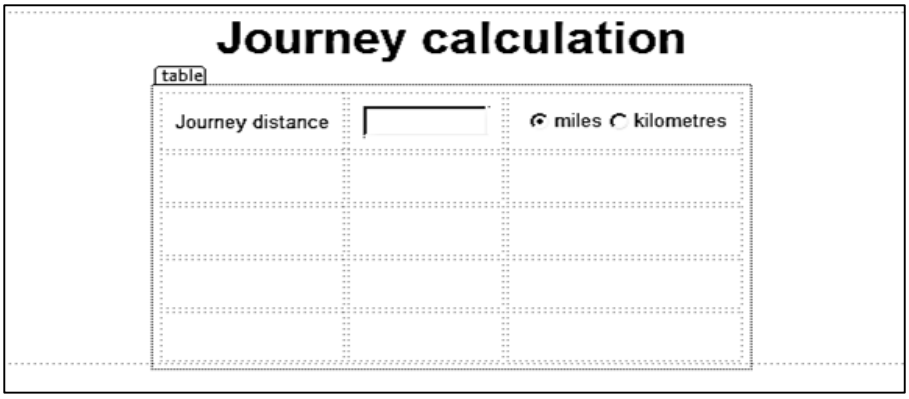


Close the web browser to return to Visual Studio. Go to the ***Debug*** option on the main menu at the top of the page and click '***Stop debugging***'.

Five rows are needed in the table for the input of data, but we have only set up code for three rows so far. Highlight the group of HTML lines from the ***<tr>*** tag to the ***</tr>*** tag which create an empty row of cells in the table. Copy this group twice to below itself.

```
<tr>
    <td>
         </td>
    <td>
         </td>
    <td>
         </td>
</tr>
```

Go to the Design window and check that the table now contains five rows of cells.



Return to the HTML page and enter code for the four other rows of the table:

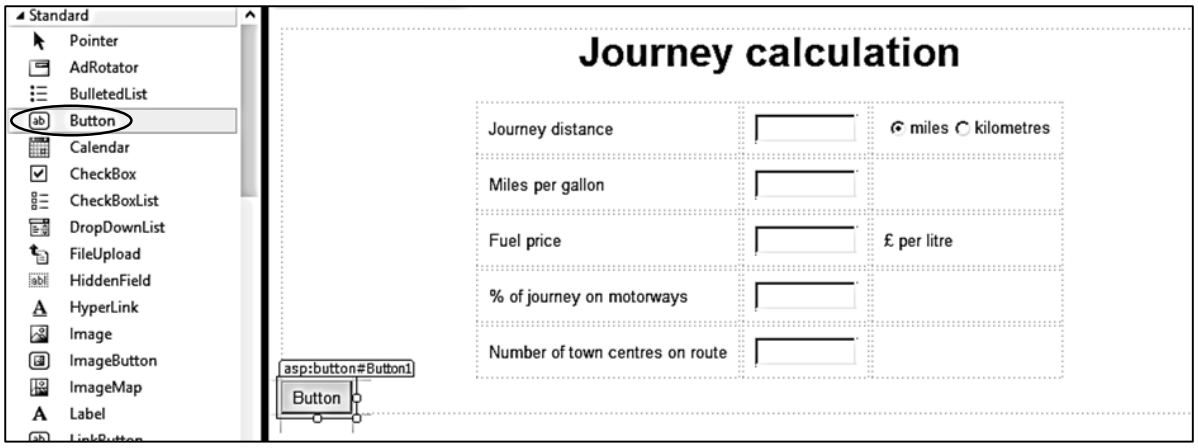
```
            <asp:RadioButton ID="rbtnKm" runat="server" GroupName="distanceUnit"
                             Text="kilometres" />
        </td>
    </tr>
    <tr>
        <td>
            Miles per gallon</td>
        <td>
            <asp:TextBox ID="txtMpg" runat="server" Width="80px"></asp:TextBox>
        </td>
    </tr>
    <tr>
        <td>
            Fuel price</td>
        <td>
            <asp:TextBox ID="txtFuelPrice" runat="server" Width="80px">
            </asp:TextBox>
        </td>
        <td>
            £ per litre</td>
    </tr>
    <tr>
        <td>
            % of journey on motorways</td>
        <td>
            <asp:TextBox ID="txtMotorways" runat="server" Width="80px">
            </asp:TextBox>
        </td>
    </tr>
    <tr>
        <td>
            Number of town centres on route</td>
        <td>
            <asp:TextBox ID="txtTowns" runat="server" Width="80px"></asp:TextBox>
        </td>
    </tr>
</table>
```

Notice that some cells in the right hand column of the table are not used, so the corresponding

```
<td>
    </td>
```

table data groups have been removed to simplify the program script. However, it will do no harm to the running of the web page if these groups are left in place.

Go to the ***Design*** window and check that all labels and text boxes are displayed correctly.
We will now add the '***Calculate***' button. Identify the ***Button*** component in the ***Toolbox***, and drag this to the web page below the table.

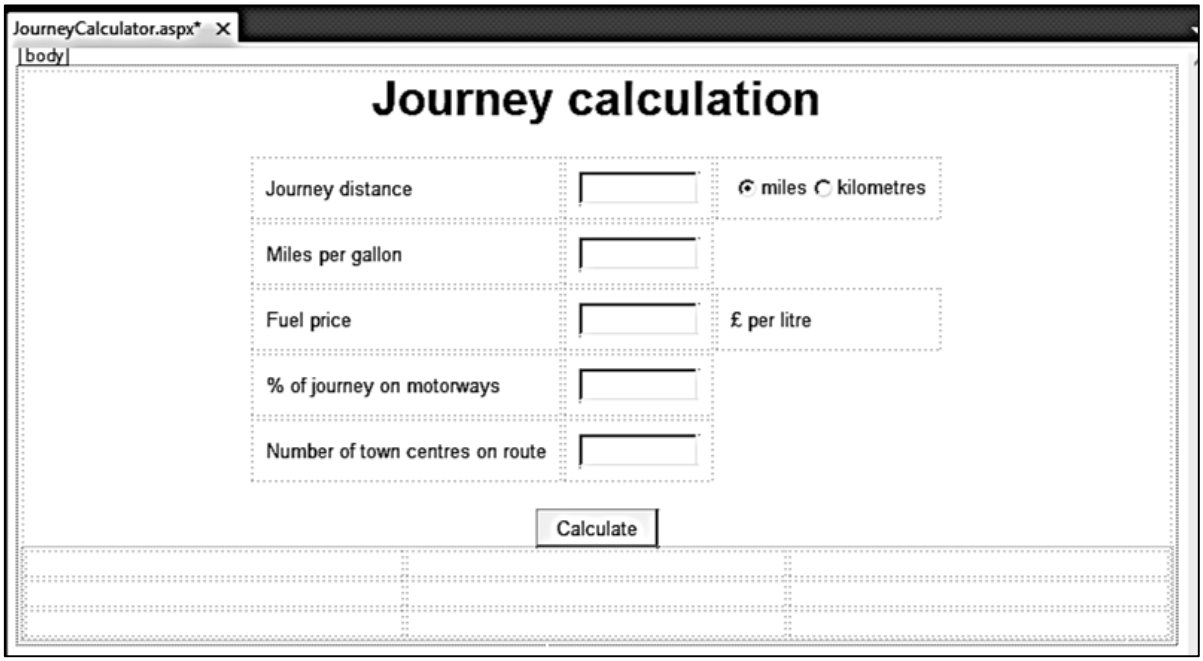


Return to the HTML page. The computer will have inserted a ***<Button>*** tag. Check that this is enclosed within the *** *** and ***</form>*** closing tags. If not, drag the ***<Button>*** line to the correct position.

We will name the button 'btnCalculate', as in the stand-alone program. Also add a *** *** tag to centre the button on the page, and ***
*** tags to produce blank lines above and below the button.

```
            <td>
                Number of town centres on route</td>
            <td>
                <asp:TextBox ID="txtTowns" runat="server" Width="80px"></asp:TextBox>
            </td>
        </tr>
    </table>

    <center>
    <br />
    <asp:Button ID="btnCalculate" runat="server" Text="Calculate" />
    <br />
    </center>

    </div>
    </form>
</body>
</html>
```

It just remains to create a set of text boxes and labels for output of the calculation results. Go to the Design window. Drag another HTML Table component onto the web page below the button.



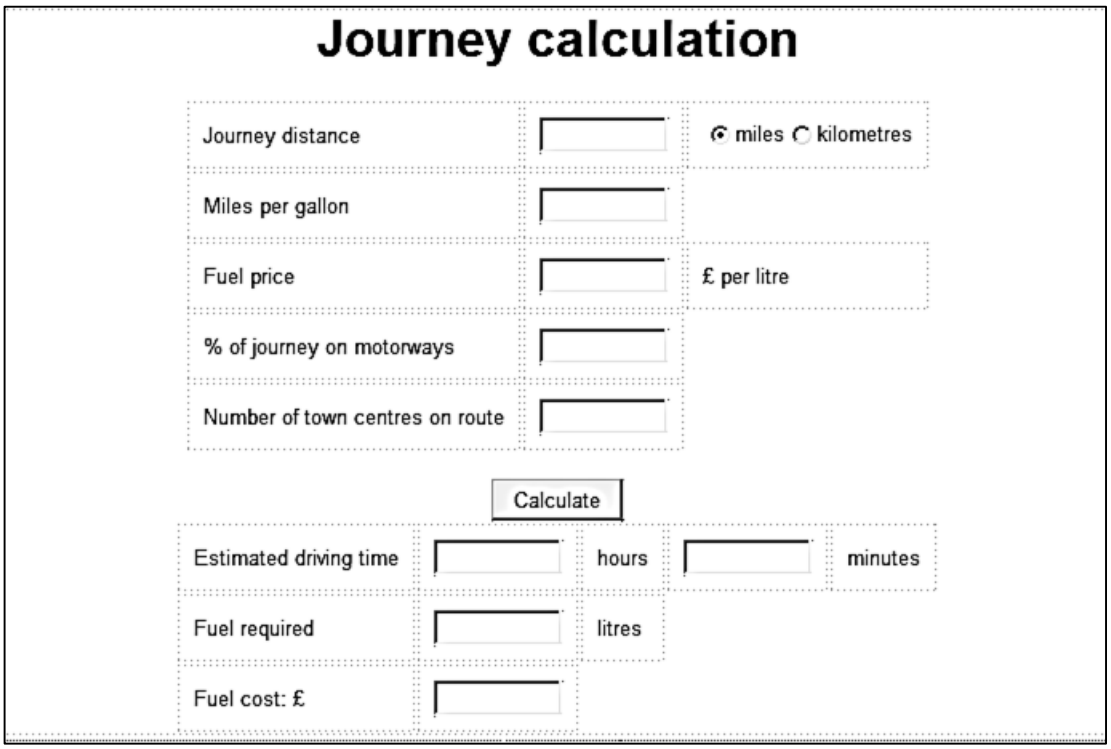
Return to the HTML page. A set of empty table rows and columns will have been added. Ensure that these are within the *** *** and ***<form>*** tags.

```
        <asp:Button ID="btnCalculate" runat="server" Text="Calculate" />
    <br />
    </center>

    <table style="width:100%;">
        <tr>
            <td> </td>
            <td> </td>
            <td> </td>
        </tr>
        <tr>
            <td> </td>
            <td> </td>
            <td> </td>
        </tr>
        <tr>
            <td> </td>
            <td> </td>
            <td> </td>
        </tr>
    </table>

    </div>
    </form>
```

Centre the table and add text boxes and captions to the table cells as shown below. We have used the same names for the text boxes as in the stand-alone program version. Notice that it is necessary to add two extra cells on the top row of the table, so that five items can be displayed.

```
<asp:Button ID="btnCalculate" runat="server" Text="Calculate" />
<br />
</center>

<table border="0" cellpadding="10" align="center">

      <tr>
         <td>
              Estimated driving time
         </td>
         <td>
              <asp:TextBox ID="txtHours" runat="server" Width="80px">
              </asp:TextBox>
         </td>
         <td>
              hours</td>
         <td>
              <asp:TextBox ID="txtMinutes" runat="server" Width="80px">
              </asp:TextBox>
         </td>

        <td>
             minutes</td>

      </tr>
      <tr>
         <td>
              Fuel required</td>
          <td>
              <asp:TextBox ID="txtLitres" runat="server" Width="80px">
              </asp:TextBox>
          </td>
          <td>
             litres</td>
      </tr>
      <tr>
          <td>
               Fuel cost: £</td>
          <td>
               <asp:TextBox ID="txtJourneyCost" runat="server" Width="80px">
               </asp:TextBox>
          </td>
      </tr>
    </table>
</div>
</form>
```

Go to the Design screen to check that captions and text boxes are displayed correctly in the output table.



We have now completed the web page interface for the application. The next step is to add the program code to carry out the calculation of journey time and fuel cost.

Double click the 'Calculate' button. The C# code page will open in exactly the same way as for the stand-alone application. Its appearance should be familiar.

```
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace Car_journey_web_application
{
    public partial class JourneyCalculator : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {

        }

        protected void btnCalculate_Click(object sender, EventArgs e)
        {

        }
    }
}
```

Locate the ***button_ click*** method. Open the ***Visual Studio*** application again as another window and load your stand-alone version of the car journey program. Copy and paste the ***button_click*** program code from the stand-alone version into the web page version.

```
protected void btnCalculate_Click(object sender, EventArgs e)
{
    try
    {
        Double distance = Convert.ToDouble(txtDistance.Text);
        if (rbtnKm.Checked == true)
        {
            distance = distance * 0.621;
        }

        Double mpg = Convert.ToDouble(txtMpg.Text);

        Double fuelRequired = distance / mpg;

        fuelRequired = fuelRequired * 4.546;

        string output = fuelRequired.ToString("f1");
        txtLitres.Text = output;

        Double fuelPrice = Convert.ToDouble(txtFuelPrice.Text);
        Double totalCost = fuelRequired * fuelPrice;

        output = totalCost.ToString("f2");
        txtJourneyCost.Text = Convert.ToString(output);

        Double motorwayPercent = Convert.ToDouble(txtMotorways.Text);
        int townCentres = Convert.ToInt16(txtTowns.Text);

        Double motorwayMiles = distance * motorwayPercent / 100;
        Double otherMiles = distance - motorwayMiles;

        Double totalMinutes = (motorwayMiles / 60 + otherMiles / 35) * 60;

        totalMinutes = totalMinutes + (townCentres * 20);

        int hours = Convert.ToInt16(totalMinutes) / 60;
        int minutes = Convert.ToInt16(totalMinutes) % 60;

        txtHours.Text = Convert.ToString(hours);
        txtMinutes.Text = Convert.ToString(minutes);
    }
    catch
    {
        MessageBox.Show("Incorrect data entered");
    }
}
```

Notice that one line of code:

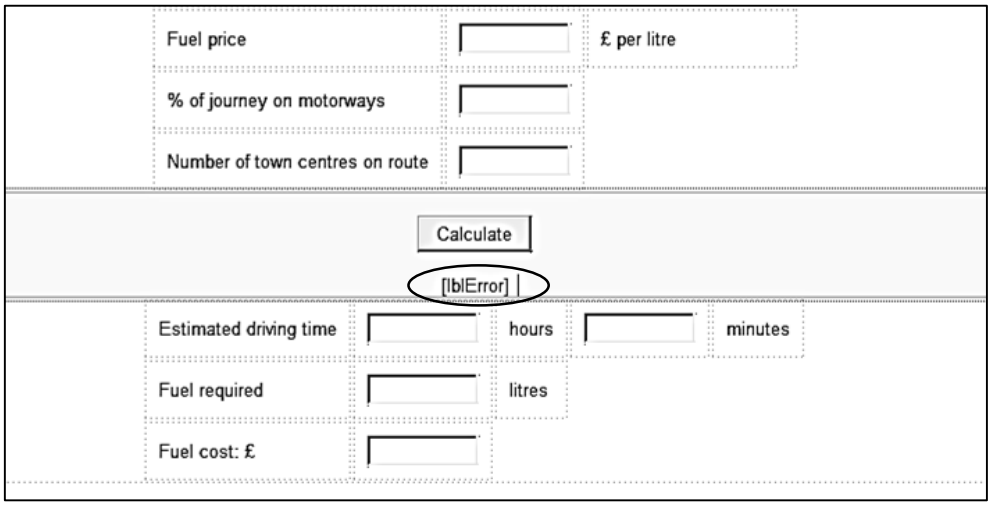
```
MessageBox.Show("Incorrect data entered");
```

has caused an error. This is because pop-up message boxes are not provided for web page applications. We can still display a message to the user, but this should be done on the web page using a ***Label*** component.

Return to the HTML code page by selecting the ***JourneyCalculator.aspx*** tab at the top of the programming window. Add a ***<Label>*** component below the ***<Button>*** code.

```
<center>
<br />
<asp:Button ID="btnCalculate" runat="server" Text="Calculate"
          onclick="btnCalculate_Click" />
<br />

<br />
<asp:Label ID="lblError" runat="server"></asp:Label>
<br />

</center>
```

Go to the Design window and check that the Label appears in the correct position below the button.



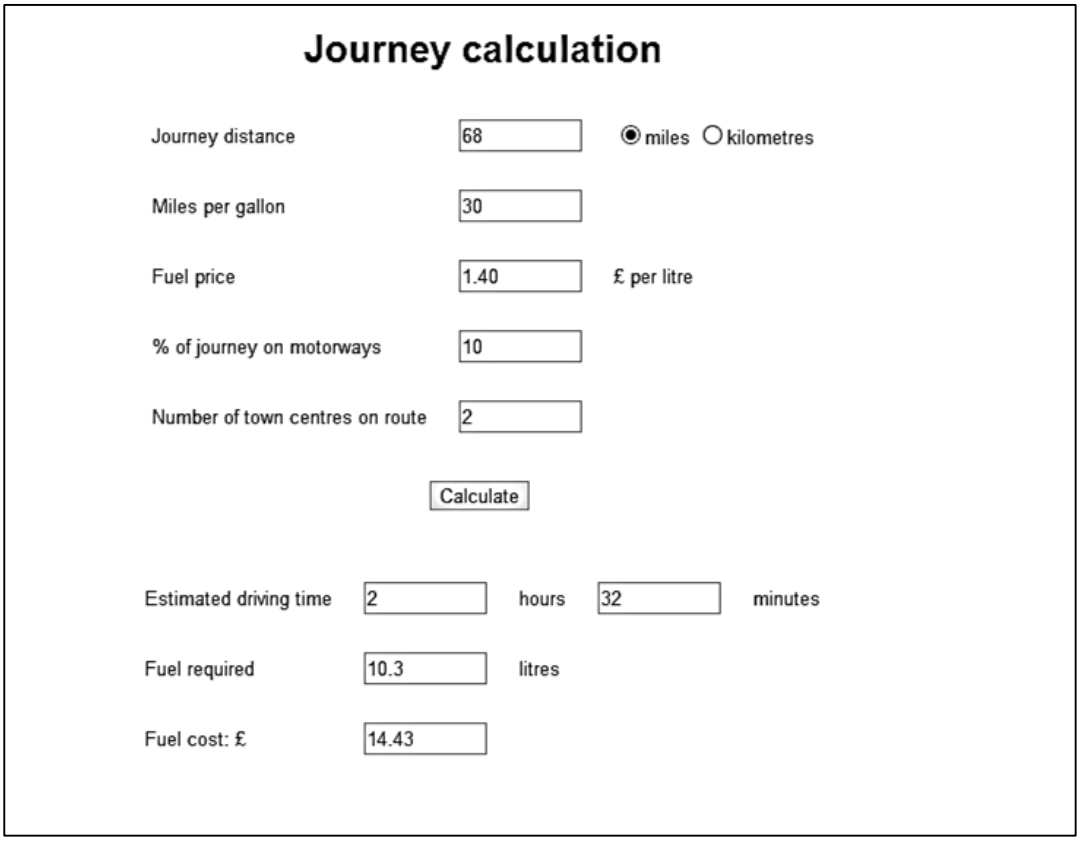
Double click the ***Calculate*** button to return to the ***button_click*** method on the C# page. Replace the line of code in the ***catch*** block.

```
    txtHours.Text = Convert.ToString(hours);
    txtMinutes.Text = Convert.ToString(minutes);
}
catch
{
    lblError.Text = "Incorrect data entered";
}
```

Add a line of code at the start of the button_click method to initialise the error label text to be blank. This will clear any error message from a previous run of the program.

```
protected void btnCalculate_Click(object sender, EventArgs e)
{
    try
    {
        lblError.Text = "";

        Double distance = Convert.ToDouble(txtDistance.Text);
        if (rbtnKm.Checked == true)
        {
            distance = distance * 0.621;
        }
```

Build and run the web page in the Internet browser. The completed program should operate in the same way as the stand-alone application.



Test the program, both for the entry of correct and incorrect data.