CMPT 110 Study Guide

Unit 11: Graphs and Simulation

Adapted from Tony Dixon’s Distance Education Version of CMPT 110 at SFU

Introduction

Objectives

- To learn how to use multiple windows in an application.
- To become familiar with the PictureBox control object and use it for displaying graphical output.
- To learn how to use the Graphics class to draw lines and other graphical objects.

Using More Than One Form

In Unit 9, you were introduced to control arrays, and it was mentioned that control objects can be created dynamically: that is, during the execution of a program. This feature is especially advantageous when a program uses multiple forms because it allows the programmer to "customize" a window from the information provided by the user in an initial "start-up" window.

Assuming you have designed your start-up window, you can create a second window as follows:

1. From the Project menu, select "Add Windows Form."
2. In the "Add New Item" dialog box click on "Windows Form."
3. Click the Add button.

A new Form is created. Since your initial (start-up) Form is likely named Form1, this Form will be named Form2. If you look in the Solution Explorer window (upper right part of the VS IDE), you should observe your two Forms. Each has its own designer and code windows. By clicking on either name, you will be able to access and display the window associated with that Form.

When one window provides the capability, through the programming of its control methods, to open a second window, it can be viewed as the "master" or control window. The window that it opens should then be designated the "servant" window. This emphasizes the role of each window.
The master window determines when the servant window should be displayed. To do so, it must be able to reference it. Since windows are represented by Visual Basic Form objects, you can assign a variable to have as its value the servant window "Form2." As with any variable declaration, you assign the value using a Dim statement:

```vbnet
Dim dataForm As New Form2
```

This statement declares that the value of the variable `dataForm` is a new instance of type `Form2`. In effect, your definition of `Form2` serves as a template for instances of `Form2` objects.

Since the master window (i.e., `Form1`) controls when the servant window (e.g., `dataForm`) gets displayed, it requires a means to do so. This is achieved with the `ShowDialog` method:

```vbnet
dataForm.ShowDialog()
```

The effect of performing this method is to give the Form, `dataForm`, the focus. During the execution of the program, the window defined by `dataForm` appears, possibly on top of the master Form. When two windows are displayed, the one with the focus appears brighter, and the one that has lost the focus appears paler.

Once the servant window has the focus, the master window is no longer accessible. To give the focus back to the master window, the servant window must be closed. This can be accomplished simply by clicking on the "X" in the top right-hand corner of the servant window. Alternatively, the window can be closed with the statement:

```vbnet
Me.Close()
```

appearing in some suitable control method, such as a `Button_Click` method, so that when the button is clicked, this statement is executed (and the window closes).

**Passing Values Between Forms**

There is one further detail that needs to be addressed in using multiple forms—the transfer of information between them. You do this using shared variables. In this case, however, the sharing takes place between forms rather than within forms.

You may have observed that until now the only shared variables were within the Class block for a Form definition. As a result, shared variables were also referred to as class-level variables and were defined in one or more Dim statements immediately following the beginning of the Class block. To share variables between forms requires that they be shared between classes, which is accomplished by declaring them to be Public when they are declared.

For example, to declare that a variable used in the servant Form is accessible to the master Form, it needs to be declared `Public` in the servant Form:
Public data_val As String

declares `data_val` to be a String variable accessible to other Forms.

Of course, you may not have designed the servant Form, and therefore may already be using the variable that the servant Form has declared Public within your Form. To avoid this possible conflict in your Form (i.e., the master Form), you need to be able to refer to distinguish Form2’s version of `data_val` from your own. You can do so by including as a prefix to the variable name, separated by a period, the name of the class where the variable is declared whenever you reference it.

Thus, if the Form dataForm declares a variable `data_val` and you wish to refer to it, you can do so as

dataForm.data_val

On the other hand, `data_val` by itself refers to the variable of that name within your own Form. Actually, all variables within your forms have an implied prefix `Me`. Thus `Me.data_val` also refers to the variable of that name within your own form. You have already been using the prefix `Me` when you wished to exit from a window.

In fact, all variables with your forms have an implied prefix `Me`. Thus `Me.data_val` also refers to the variable of that name within your own form. You have already been using the prefix `Me` when you wished to exit from a window with `Me.Close()`.
An Example Using Multiple Forms

All the essential ideas described for transferring values between Forms can be illustrated in the following simple example:

**Problem Description**

Design an input interface where the master window displays data obtained from a servant window. Data entry is only possible through the servant window which is accessed using a button.

**Requirements Analysis**

**Input Requirements**

<table>
<thead>
<tr>
<th>INPUT</th>
<th>TYPE</th>
<th>PURPOSE</th>
<th>REPRESENTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_value</td>
<td>String</td>
<td>data input field</td>
<td>TextBox (Form 2)</td>
</tr>
</tbody>
</table>

**Output Requirements**

<table>
<thead>
<tr>
<th>OUTPUT</th>
<th>TYPE</th>
<th>PURPOSE</th>
<th>REPRESENTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>data_value</td>
<td>String</td>
<td>data output field</td>
<td>TextBox (Form 1)</td>
</tr>
</tbody>
</table>

**Event Requirements**

<table>
<thead>
<tr>
<th>EVENT</th>
<th>EFFECT</th>
<th>REPRESENTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>get data</td>
<td>open Form 2</td>
<td>Button</td>
</tr>
</tbody>
</table>
Form Layouts

Implementation

```vbnet
Public Class Form1

Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    ' data_value declared Public in Class Form2
    Dim dataForm As New Form2
    dataForm.ShowDialog()
    TextBox1.Text = dataForm.data_value
End Sub
End Class
```

```vbnet
Public Class Form2

Public data_value As String

Private Sub TextBox1_TextChanged(sender As Object, e As EventArgs) Handles TextBox1.TextChanged
    data_value = TextBox1.Text
End Sub
End Class
```

Program Analysis

When execution begins, only Form 1 is displayed, with the output TextBox disabled to prevent entering data. Clicking on the "GET DATA" button opens Form 2. Text can be typed into the TextBox shown in Form 2. When the data is ready to be displayed in Form 1, the window showing Form 2 is closed by clicking on the "X" in the upper right hand corner. Form 1 now displays in its TextBox the data that was entered in Form 2.
PictureBox Control Objects

Like all contemporary programming languages, Visual Basic provides the means for programmers to write programs whose output is graphic rather than textual. For this purpose, you use a particular control object called a **PictureBox**. Within the **PictureBox** it is possible to display images that include line drawings as well as photographs that have been stored digitally. **PictureBox** objects are found in the VS IDE ToolBox and can be placed in a Form in much the same way as any other control, such as a Button or a **TextBox**.

A picture is stored in computer memory as a two-dimensional array of picture elements, called "pixels." Each pixel represents a shade of grey (if the image is black-and-white) or a colour. Actually, a colour is specified by not one but three values since every shade can be defined by a combination of the colours red, green, and blue.

Although each pixel defines a single colour, that colour occupies only a small portion of the entire image, and as a consequence, much like a newspaper or magazine image that is composed of coloured dots, an image is observed when the two-dimensional array of pixels is viewed from even a short distance.

To write programs that produce graphics images, therefore, the programmer must be able to assign values to each of the pixels of the **PictureBox** according to what colour is required at each point. Since it is necessary to be able to specify a specific pixel location, that location is determined by a row position and a column position in much the same way that an element in a table was determined in Unit 9.

The horizontal position, sometimes called the "x-position" is a value between 0 and the actual width of the picture box, which is measured in pixel widths. X-position 0 is at the extreme left of the picture box.

Similarly the vertical position, referred to as the "y-position" is also a value between 0 and the actual height of the picture box, measured in pixel heights. Unlike the mathematical Cartesian co-ordinate system, however, y-position 0 is at the extreme top of the picture box. This is an important fact to remember if you wish to display the graph of a mathematical function. To do so, you will need to convert between the picture box co-ordinate system where the origin is in the top left corner (with y increasing in value "down" the screen) and the Cartesian co-ordinate system where the origin may be anywhere and y increases in value from the bottom to the top of the plane. As well, pixel positions are non-negative integers, while each co-ordinate value in the Cartesian plane is a Real number that can be either positive or negative.

Although pixels are referenced by a row and column position, a **PictureBox** is not a two-dimensional array. Therefore, pixels in a picture box cannot be accessed with a pair of subscripts. Instead, the Visual Basic library provides a set of methods for constructing graphical images. Many of these methods include arguments that specify the x-position and the y-position of a point in the **PictureBox**.
Graphic Elements as Data Objects

One of the applications of PictureBoxes is to provide a way of displaying graphics objects. The Visual Basic library provides methods that act on instances of a specific class of abstract objects called the "graphics class." When created, instances of the graphics class are associated with a specific PictureBox.

To create a new graphics object, the Dim statement is used:

```vbnet
Dim gr As Graphics = picBox.CreateGraphics
```

This statement declares that the variable gr has type Graphics and that its value is a graphics object associated with the picture box named picBox. Of course, you could also have achieved the same effect with the following two lines:

```vbnet
Dim gr As Graphics
gr = picBox.CreateGraphics
```

Following either way of creating a graphics object, all the graphics methods are then applied to the value of gr. The number of methods available is very large. Some of the methods are described in the textbook. For a complete list:

1. Click on the Object Browser tab near the top of the VS IDE.
2. Enter "Graphics" in the search text field, and click on "Search" (the green arrow to the right of the search text field).
3. In the list below the search field, click on "System.Drawing.Graphics." To the right will be displayed a list of the methods associated with Graphics objects.
4. Click on any one of the methods. Below the list will be displayed the format of the method (i.e., the method name and formal parameters) and a description of the purpose of the method.

The following example illustrates the concepts and features of Visual Basic introduced in this and the previous unit. In particular, it demonstrates the use of files, multiple windows, and graphics objects in a single application.

Problem Description
A general purpose graphing utility is to be designed that permits the user to graph a set of points obtained from a file and display them in a separate window. From information provided initially by the user, the scale to be used in the display is determined. This information is also used to plot the x-axis and y-axis in the display in the appropriate positions. Specifically, the user will indicate the minimum and maximum co-ordinate positions that should be visible in the display. These will correspond to the top, bottom, left, and right margins of the display area, a PictureBox.

The Form that displays the graph is to be a servant window to the master Form. Servant windows will allow the user to create graphs with different sizes and with the co-ordinate axes in different positions. By exiting from the graph Form, the user can return to the master Form and enter a new set of values for the boundaries of the PictureBox.

To plot the data, the graph Form will open a third window through which the user will
provide the name of a file where a list of points, each expressed as a pair of values and stored on a single line of the file, is found. Once the data has been read, the window will be closed and the set of points plotted. This window will be a servant to the graph Form. The requirements definition for each window and the source code for each Form are given on the this and the following pages.

**An Example Using Graphs and Files (Part 1)**

The following example illustrates the concepts and features of Visual Basic introduced in this and the previous unit. In particular, it demonstrates the use of files, multiple windows, and graphics objects in a single application.

**Problem Description**

A general purpose graphing utility is to be designed that permits the user to graph a set of points obtained from a file and display them in a separate window.

From information provided initially by the user, the scale to be used in the display is determined. This information is also used to plot the x-axis and y-axis in the display in the appropriate positions. Specifically, the user will indicate the minimum and maximum co-ordinate positions that should be visible in the display. These will correspond to the top, bottom, left, and right margins of the display area, a **PictureBox**.

The Form that displays the graph is to be a servant window to the master Form. Servant windows will allow the user to create graphs with different sizes and with the co-ordinate axes in different positions. By exiting from the graph Form, the user can return to the master Form and enter a new set of values for the boundaries of the **PictureBox**.

To plot the data, the graph Form will open a third window through which the user will provide the name of a file where a list of points, each expressed as a pair of values and stored on a single line of the file, is found. Once the data has been read, the window will be closed and the set of points plotted. This window will be a servant to the graph Form.

The requirements definition of each window and the source code for each Form are given on the this and the following pages.
**Requirements Definition**

### Input Requirements

<table>
<thead>
<tr>
<th>INPUT</th>
<th>TYPE</th>
<th>PURPOSE</th>
<th>REPRESENTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmin</td>
<td>Double</td>
<td>Left margin position</td>
<td>TextBox (Form 1)</td>
</tr>
<tr>
<td>xmax</td>
<td>Double</td>
<td>Right margin position</td>
<td>TextBox (Form 1)</td>
</tr>
<tr>
<td>ymin</td>
<td>Double</td>
<td>Bottom margin position</td>
<td>TextBox (Form 1)</td>
</tr>
<tr>
<td>ymax</td>
<td>Double</td>
<td>Top margin position</td>
<td>TextBox (Form 1)</td>
</tr>
<tr>
<td>data_file</td>
<td>String</td>
<td>File name &amp; path</td>
<td>TextBox (Form 3)</td>
</tr>
</tbody>
</table>

### Output Requirements

<table>
<thead>
<tr>
<th>OUTPUT</th>
<th>TYPE</th>
<th>PURPOSE</th>
<th>REPRESENTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>gr</td>
<td>Graphics</td>
<td>Display the graph</td>
<td>PictureBox (Form 2)</td>
</tr>
<tr>
<td>message</td>
<td>String</td>
<td>Display messages</td>
<td>TextBox (Form 3)</td>
</tr>
</tbody>
</table>

### Event Requirements

<table>
<thead>
<tr>
<th>EVENT</th>
<th>EFFECT</th>
<th>REPRESENTATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRAPH</td>
<td>Create a coordinate plane</td>
<td>Button (Form 1)</td>
</tr>
<tr>
<td>SHOW AXES</td>
<td>Draw axes on coordinate plane</td>
<td>Button (Form 2)</td>
</tr>
<tr>
<td>PLOT DATA</td>
<td>Draw a graph of the data</td>
<td>Button (Form 2)</td>
</tr>
<tr>
<td>READ FILE</td>
<td>Get data to be plotted from file</td>
<td>Button (Form 3)</td>
</tr>
<tr>
<td>EXIT</td>
<td>Close Form 3</td>
<td>Button (Form 3)</td>
</tr>
<tr>
<td>EXIT</td>
<td>Close Form 2</td>
<td>Button (Form 2)</td>
</tr>
</tbody>
</table>
Notice that the requirements definition only identifies the inputs, outputs, and events. Whether the corresponding objects are placed on a single window or multiple windows is not determined until the Forms are designed. As part of the Forms design, it will be necessary to decide which objects are placed in which windows.

**Hierarchy Diagram**

To assist in the forms design it is helpful to sketch out the tasks that will need to be performed. An hierarchy diagram can be used to represent how the program will need to be organized:

This hierarchy diagram suggests defining three forms:

1. A Form will be used to get the data necessary to define the coordinate plane. Therefore the inputs: \( \text{xmin}, \text{xmax}, \text{ymin} \) and \( \text{ymax} \) will be part of this form.
2. A Form will be used to get the data necessary to retrieve the coordinate pairs from a file. Therefore the input \( \text{data_file} \) will be part of this form.
3. A Form will be used to display the graph. Therefore the \text{Graphics} object, \( \text{gr} \), will be output on this form.
Defining the Coordinate Plane (Part 2)

Form 1: The Master Window

Form 1: Visual Basic Source Code

```vbnet
Public Class Form1
    ' The PictureBox boundaries are shared by all forms.
    Public xmin As Double 'Left Margin
    Public ymin As Double 'Bottom Margin
    Public xmax As Double 'Right Margin
    Public ymax As Double 'Top Margin

    ' Button1 (GRAPH) loads the text values into the boundary variables:
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
        Dim graphForm As New Form2
        Dim xMin = CDbl(TextBox1.Text)
        Dim yMin = CDbl(TextBox2.Text)
        Dim xMax = CDbl(TextBox3.Text)
        Dim yMax = CDbl(TextBox4.Text)
        graphForm.ShowDialog()
    End Sub

    ' Button2 (QUIT) terminates execution of the program
    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button2.Click
        Me.Close()
    End Sub
End Class
```
Displaying the Graph (Part 3)

Form 2: The graphing window

Form 2: Visual Basic Source Code (next page)
Public Class Form2
  ' Class level variables shared by graphing functions:
  Dim gr As Graphics
  Dim picWidth As Integer 'the width of the PictureBox (in pixels)
  Dim picHeight As Integer 'the height of the PictureBox (in pixels)
  Dim xmin As Double, ymin As Double, xmax As Double, ymax As Double

  ' DefineBoundaries sets the class-level variables according
  ' to the values assigned to the Form level equivalents:
  Sub DefineBoundaries()
    xmin = Form1.xmin
    xmax = Form1.xmax
    ymin = Form1.ymin
    ymax = Form1.ymax
  End Sub

  Sub DrawGraphAxes()
    gr.DrawLine(Pens.Blue, Xposn(xmin), Yposn(0), Xposn(xmax), Yposn(0)) ' X-axis
    gr.DrawLine(Pens.Blue, Xposn(0), Yposn(ymin), Xposn(0), Yposn(ymax)) ' Y-axis
  End Sub

  ' Xposn converts an x-coordinate to a pixel column position
  Function Xposn(ByVal xVal As Double) As Integer
    Return CInt((xVal - xmin) / (xmax - xmin) * CDbl(picWidth))
  End Function

  ' Yposn converts a y-coordinate to a pixel row position
  Function Yposn(ByVal yVal As Double) As Integer
    Return CInt((yVal - ymin) / (ymax - ymin) * CDbl(picHeight))
  End Function

  ' DrawGraph retrieves each point and plots it as a circle of radius 2 in the PictureBox.
  ' Form2 provides the points as shared variables:
  ' fileSize gives the number of points to plot
  ' point(n) gives the nth point as a string of values separated by a comma.
  ' "Split" separates them into individual values.
  Sub DrawGraph(ByVal ByVal form As Form3)
    Dim coords() As String
    Dim x, y, radius As Double
    Const radius As Integer = 2
    For n As Integer = 1 To form.fileSize
      coords = form.point(n).Split("",";")
      x = CDbl(coords(0))
      y = CDbl(coords(1))
      gr.FillEllipse(Brushes.Black, Xposn(x) - radius, Yposn(y) - radius, 2 * radius, 2 * radius)
    Next n
  End Sub

  ' Button2 (SHOW AXES) draws the x-axis and y-axis of the graph:
  Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles Button2.Click
    gr.Clear(Me.BackColor) ' refresh the background
    ' color of Form2
    DrawGraphAxes()
  End Sub

  ' Button3 (PLOT DATA) Obtains the data and plots it on the graph:
  Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles Button3.Click
    Dim getdataForm As New Form3
    getdataForm.ShowDialog()
    DrawGraph(getdataForm)
  End Sub

  ' Button4 (EXIT) Closes graph window, returning to Form1
  Private Sub Button4_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles Button4.Click
    Me.Close()
  End Sub

  Private Sub Form2_Load(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) _
    Handles MyBase.Load
    gr = PictureBox1.CreateGraphics
    DefineBoundaries()
    picWidth = PictureBox1.Width
    picHeight = PictureBox1.Height
  End Sub
End Class
Retrieving the Points to Plot (Part 4)

Form 3: The "Data Retrieval" Window

Form 3: Visual Basic Source Code

```vbnet
Public Class Form3
    Public point(1000) As String 'Up to 1000 points read from the file and stored in the array "point".
    Public fileSize As Integer 'The actual number of points is given by "fileSize".

    Private Sub Form3_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) _
        Handles MyBase.Load
        Textbox1.Text = "" ""
    End Sub

    'Button1 (READ FILE) retrieves the points from the file provided in Textbox1 and copies each line 'to the array "point". A line consists of a pair of values separated by a comma.
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
        Handles Button1.Click
        Dim rdr As IO.StreamReader
        If IO.File.Exists(Textbox1.Text) Then
            rdr = IO.File.OpenText(Textbox1.Text)
            fileSize = 0
            Do Until rdr.EndOfFile
                fileSize = fileSize + 1
                point(fileSize) = rdr.ReadLine
            Loop
            Textbox2.Text = "Data file " & Textbox1.Text & " has been read."
            rdr.Close()
        Else
            Textbox2.Text = "File Not Found"
        End If
    End Sub

    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) _
        Handles Button2.Click
        Me.Close()
    End Sub
End Class
```
Executing the Graphing Program (Part 5)

To demonstrate the execution of the program, suppose that the following pairs of points are to be plotted:

(-10, 8), (-9, 7), (-8, 6), (-7, 5), (-6, 4), (-5, 3), (-4, 2), (-3, 1), (-2, 0), (-1, 1), (0, 2), (1, 3), (2, 4), (3, 5), (4, 6), (5, 7), (6, 8), (7, 9), (8, 10)

This data can be stored in a file using a basic editor such as NotePad (PC). Each coordinate pair is stored in a separate line of the file in CSV format (i.e., Comma Separated Values). The first few lines of the file would be:

-10, 8
-9, 7
-8, 6

Assume the file is stored at "U:\CMPT110\data.txt"

When the program is launched Form 1 appears. The minimum and maximum values of the x-axis and the y-axis must be specified. In this case the x-axis is chosen to range from -20 to 30, and the y-axis is chosen to range from -5 to 30:
When the axes have been defined, "GRAPH" is clicked, and Form 2 is opened. It consists of an empty form except for three buttons: "SHOW AXES", "PLOT DATA", and "EXIT." By clicking "SHOW AXES" the x-axis and y-axis specified in Form 1 are displayed:

By clicking the "PLOT DATA" button, Form 3 is launched:
The directory path and file name are entered and the "READ FILE" button is clicked. The Read Only [TextBox] indicates whether the file is successfully read. When the "EXIT" button is clicked, Form 2 is reopened with the points read from the file plotted on the graph:

![Form 2 graph](image)

By clicking on the "PLOT DATA" button a second time, a second set of points can be read and plotted on the same graph as the first set of points. Otherwise, by clicking on "EXIT," the program returns to Form 1, so that a new set of axes may be specified and the second set of data plotted on a new graph.