LAB 2: Writing Simple Visual Applications

Duration : 2 Hours

Learning Outcomes

This Labsheet encompasses of activity 2A,2B, 2C and 2D

By the end of this laboratory session, you should be able to:

- Add a toolbar to a Form
- Create multiple Forms in VB. NET.
- Use suitable Object Controls: Shapes, Textbox, Label, Command Button, List Box, Combo Box, Option Button, Check Box, Frame, Scroll Box, Frame Controls and others
- Set the properties of Object Control
- Open the Code Window for that particular object.
- Identify the Event Sequence.
- Write Event Procedures for the object.

Hardware/Software: Visual Studio with latest version

Activity Outcome:

By the end of this lab, students should be able to :

Theory/ Topics

The first thing to note about visual applications is that they are event-driven. This means that the application appears to do nothing until "something happens" (ie an event occurs) then a piece of code is executed before the application appears to go back to doing nothing. Different pieces of code are executed for different events. An event can be something like a button click, the mouse pointer entering or leaving a control, a key up or down, etc. Thus, at first sight a visual application seems to be a collection of separate little bits of code.

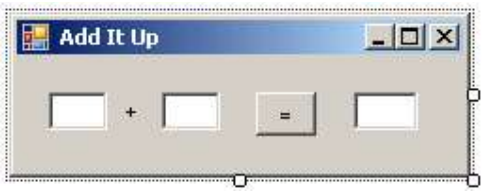
Activity 2A

Activity Outcome: Creating an interface to add two numbers.

Building the Adder Example Program

Procedure :

Step 1 - Design the User Interface



Start a new Visual Basic .NET project in the manner described in the previous lab. Design a form with three TextBoxes, a Button, and a Label that looks like that shown. This will be the user interface for a simple adder, ie the sum of the values in the two leftmost TextBoxes will be put into the rightmost TextBox whenever the Button is clicked.

Step 2 - Name the All the Controls

This is still a very small application, nevertheless it has three TextBoxes. The default names for these TextBoxes are **TextBox1, TextBox2,** and **TextBox3**. Which one is which depends on the order in which the TextBoxes were placed on the form. Which TextBox contains the sum? You should see that for larger applications we are going to get into a serious mess. The solution to this is to rename all the controls meaningfully.

A meaningful name for a control should indicate what kind of control it is and what it does. The convention is to use a prefix to specify the type of control, so a meaningful name for the first TextBox would be **txtFirst**. Note the capitalisation of the initial letter of the internal word First, this is the convention in Visual Basic to make names easier to read in the absence of spaces which are not allowed (similar to Pascal). Here the prefix **txt** stands for "TextBox". A list of the prefixes conventionally used for the standard controls is shown. Thus name the TextBoxes **txtFirst, txtSecond,** and **txtSum**, and the button **cmdAdd**. The Label control will not be referred to in code so can keep its default name.

Control	Prefix
ComboBox	cb
CommonDialog	cdc
CheckBox	chk
CommandButton	cmd, btn
DriveListBox	drv
DirListBox	dir
FileListBox	fil
FlexGrid	flx
Frame	grp
Image	img
ImageList	imgl
Label	lbl
ListBox	lst
ListView	lv
MaskedEdit	msk
MainMenu	mnu
PictureBox	pb
ProgressMeter	prg
RadioButton	rdb
StatusBar	sbr
ScrollBar	scr
Timer	tim
Toolbar	tlb
TreeView	tv
TextBox	txt

Note that the renaming of controls has to be done before any event handlers are written for them.

Step 3 - Write the Event Handler

This is a very simple application. The only time we want anything to happen is when the button is clicked. In other words - the only event that needs to be handled is the button click. Thus all the code for this application will be in the button click event handler.

Double click on the button to open its click event handler as shown:

```
Public Class Form1

    Private Sub cmdAdd_Click(ByVal sender As System.Object, ByV

    End Sub
End Class
```

Try entering this single line of code inside the event handler:

```
txtSum.Text = txtFirst.Text + txtSecond.Text
```

Run the application and try it out with a simple sum (use integer values). Do you get the expected answer? What you actually get is:

It's actually concatenating the two numbers rather than adding them! The "+" operator is overloaded, ie it does different things depending on the types of its operands (the operands of the "+" operator are the values that come before and after it). If the operands are of any arithmetic type then the "+" operator will perform an addition on them. However if the operands are strings (the .Text property is a string) it performs a concatenation. Note the concatenation operator is "&" but "+" works as well. It's probably best to use "&" to show explicitly that you mean concatenate rather than add. This may help when doing code maintenance.

So how do we make the application ADD the two numbers? Replace the line of code from above with the three below:

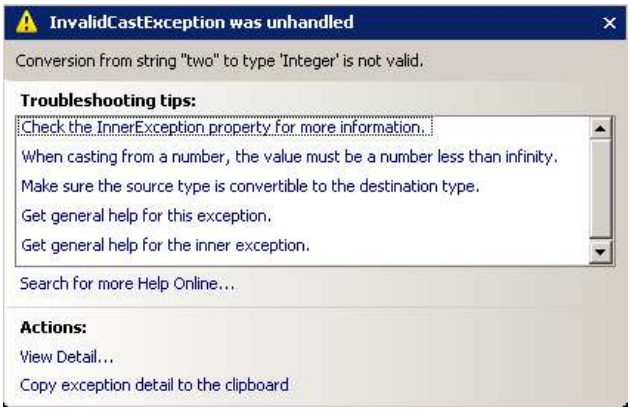
```
Dim intFirst As Integer = txtFirst.Text
Dim intSecond As Integer = txtSecond.Text
txtSum.Text = intFirst + intSecond
```

In line 1 a local variable of type Integer is declared and initialised with the value of the Text property of the first TextBox. The Text property is a string but is automatically converted to a integer on the assignement.

Similarly in line 2 for the second TextBox.

Line 3 uses the "+" operator with two integer parameters hence it performs an add. The sum is then assigned to the Text property of the sum TextBox. The sum is automatically converted from an integer to a string on the assignment. The following shows this working:

The code we have just written isn't idiot-proof. It assumes the user will enter integer values into the first two TextBoxes (a big assumption); what if she doesn't?

User Enters Floating Point Numbers: When the two strings are converted to integers they are rounded, so the sum is 3 + 6 which gives 9.	
User Enters Non-Numeric Text: The non-numeric text "two" cannot be automatically converted to an integer. When an attempt to do so is made in the code a run-time error occurs. The dialog shown appears with information that is useful to you the programmer. However this information will mean nothing to an average user: they will just be very annoyed that the application that they were using has just crashed. We will do more on run-time errors in later weeks!	 

Prefix	Data Type
ar	Array
by	Byte
cur	Currency
dbl	Double
dec	Decimal
dt	Date
fl	Boolean
int	Integer

You will often see a naming convention for variables where a type specific prefix is put on the variable name. In the above example the local variables were integers so the prefix "int" was added to indicate this. Beware, using such a prefix is for documentation only and doesn't affect the type of a variable. A list of the common prefix's used is shown. This convention is less rigorously applied than that for the names of controls.

lng	long
obj	Object
sng	Single
str	String
var	Variant

Activity 2B

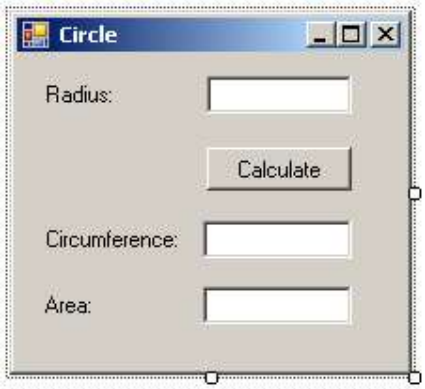
Activity Outcome: Creating an interface to calculate the circumference and area of a circle

Building the Circle Example Program

Procedure :

Step 1 - Design the User Interface

Start a new Visual Basic .NET project in the manner described in the previous lab. Design a form similar to that shown. Give the TextBoxes the meaningful names: txtRadius, txtCircum, txtArea, and give the button the name: cmdCalculate.



Step 2 - Define a Constant

Use **View | Code** to bring up the code view and define a symbolic constant for pi:

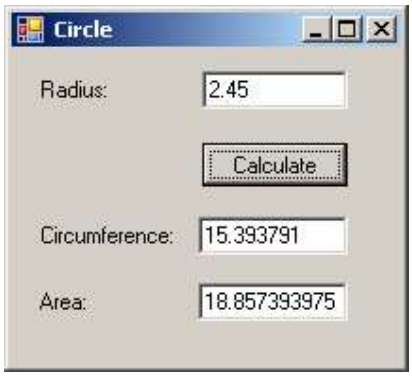
```
Public Class Form1
    Private Const PI = 3.14159
End Class
```

Step 3 - Code the Button Click Event Handler

Some very simple code for the button click event handler is shown:

```
    Private Sub cmdCalculate_Click(ByVal sender As Object, ByV
        Dim dblRadius As Double = txtRadius.Text
        txtCircum.Text = 2 * PI * dblRadius
        txtArea.Text = PI * dblRadius ^ 2
    End Sub
```

This will result in the user interface looking like:



The first line takes the (hopefully) numeric string in the radius TextBox and automatically converts it to the floating-point type double. The next two lines do the calculations for the circumference and the area respectively. In both these lines a value of type double is automatically converted to a string and displayed in the TextBoxes. The answers shown are correct but are shown to too many decimal places to be easily read. You will see how to format a string to a specific number of decimal places when we look at the String class in detail.

Sequence of Statements

Within a piece of code (eg an event handler) the individual statements are normally executed in the order that they are listed. This is the default flow of control within a program. To change this there are also "control structures" for decision (selection) and for repetition (iteration). These are special statements that alter the flow of control. We will look at these in detail in the following two labs. This next example shows how the sequence of statements is important.