An Introduction to Visual Basic Programming

Table of Contents: page

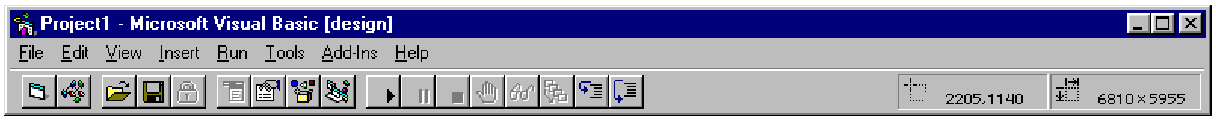
An Introduction to Visual Basic Programming

Introduction

This course is designed as an introduction to programming using Visual Basic. For many people, this will be a new and powerful environment and, while it might take a bit of getting used to at first, the advantages and flexibility in design that it offers should soon become apparent. We also think that learning shouldn't always be deadly serious and so we'll be disappointed if you don't have some fun along the way....so, let's get started right now!

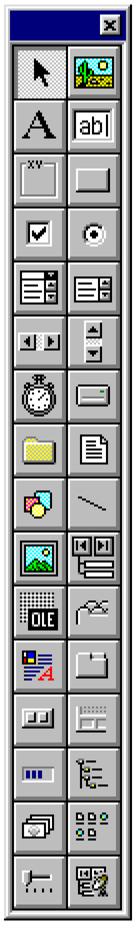
Running Visual Basic

Your system should be set up so that you can run Visual Basic from the program menu. For a typical installation, several windows should appear when you do this. The one at the top of the screen is the most important and looks like this:


Project1 - Microsoft Visual Basic [design]
File Edit View Insert Run Tools Add-Ins Help
2205,1140
6810 x 5955

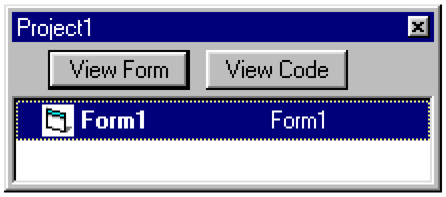
This is the main menu and toolbar. We can see that the default name for the 'new' project is **Project1** and we also notice that the title bar indicates that we are in *design* mode.

Also visible should be the toolbox. If it isn't, then click **View** on the menu bar then **Toolbox** on the drop-down menu. It should look something like this, but it may just have one or two more, or less, buttons on it:

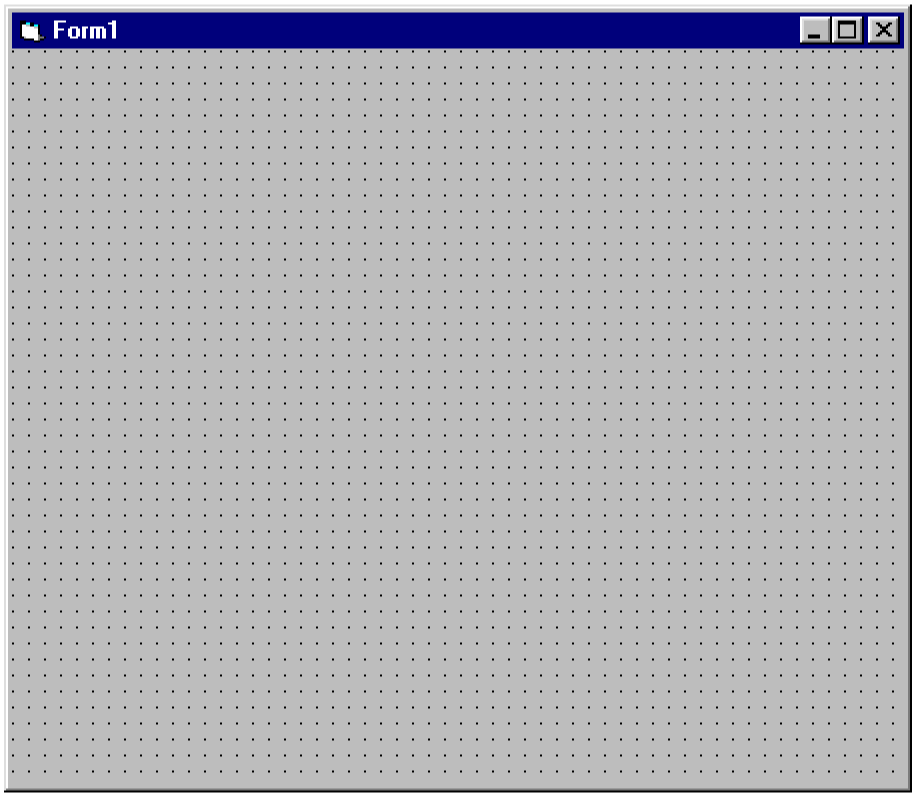


Each of the buttons represents a *control* and if you hold the mouse over any of them, a small box should appear indicating which control it is. The one at the top right, for example, should say 'Picture Box'. These small labels are called 'Tool Tips' and if you don't see them then click **Tools** on the menu bar then **Options** and in the dialog box that appears, make sure that the **Show Tool Tips** box is ticked - it should be under the **Environment** tab.

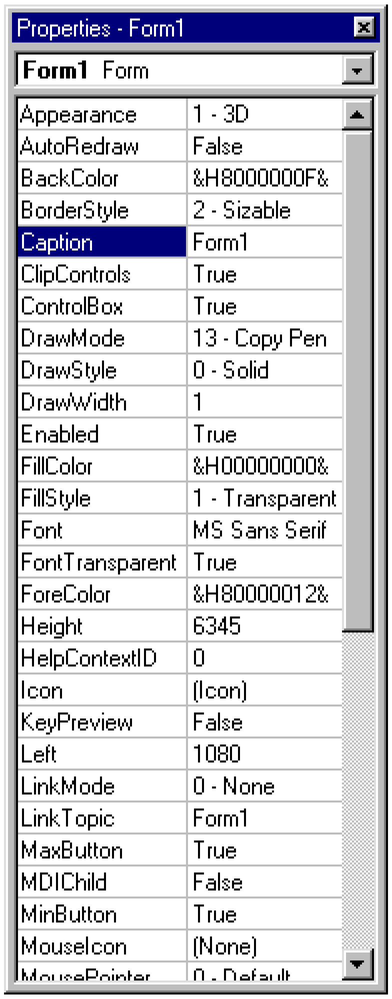
There should be a small 'Project' window similar to this:



If you can't see the 'Form' window, then click View Form. It looks like this:



Finally, there is a 'Properties' window where you will set *properties* for the various *controls* that your projects will use. Right now, your project has only one control - namely, the form - that you can see. The window looks like this:



Appearance	1 - 3D
AutoRedraw	False
BackColor	&H8000000F&
BorderStyle	2 - Sizable
Caption	Form1
ClipControls	True
ControlBox	True
DrawMode	13 - Copy Pen
DrawStyle	0 - Solid
DrawWidth	1
Enabled	True
FillColor	&H00000000&
FillStyle	1 - Transparent
Font	MS Sans Serif
FontTransparent	True
ForeColor	&H80000012&
Height	6345
HelpContextID	0
Icon	(Icon)
KeyPreview	False
Left	1080
LinkMode	0 - None
LinkTopic	Form1
MaxButton	True
MDIChild	False
MinButton	True
MouseIcon	(None)
MousePointer	0 - Default

At the moment this is called **Form1**, but you can use the 'Properties' window to change the name of the form and, indeed, to change any of the other properties that you can see listed - the background colour is just one of several properties that affect the *appearance* of the form. You can also alter the *size* and *position* of the form on the screen. In fact you can do this in two ways; the first is by typing in numbers to specify Height, Width and so on, and the second is by using the mouse to reposition or resize the form which will then automatically alter the numbers in those properties - try it and see.

The numbers used to specify the height, width and position on the screen are measured in *twips*. What was that? Twips? Never heard of them, right? Don't worry. It's possible to set your system to use other units but we're going to stay with twips throughout this course. If you write programs that might be used on other systems which may have different monitors and/or screen resolutions then you'd need to give some consideration to this issue but it won't concern us here.

If you look near the top right hand corner of the title window, you'll see two small images with pairs of numbers, or *coordinates*, beside them. These tell you the size of the form and its position on the screen. If you resize the form or reposition it, you'll see how the numbers change. The position given by the coordinates, is the position of the top left corner of the form and it is measured from the top left corner of the screen.

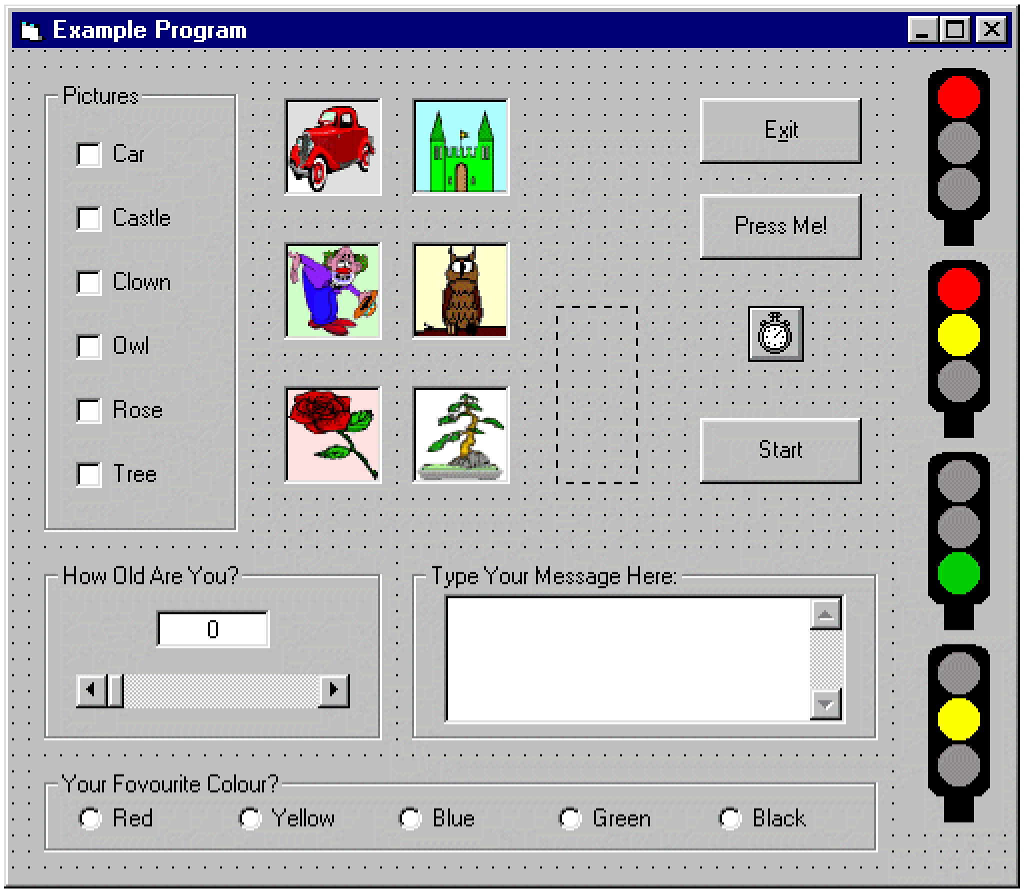
That's given you a very quick tour of what Visual Basic looks like when you run it. You've probably got a lot of unanswered questions, but at his stage, it would be a little odd if you didn't have - start worrying when you stop asking questions!

An Example Program

Before we start programming ourselves, let's load up an example program just to see if we can pick out a few of the things that you might be looking out for. The course materials come with several image files and also this sample project. Where you find it will depend on how your system is set up - typically it will be in a directory or folder called **projects** and you're looking for one called **example.vbp**. This will probably be the directory where you're going to save your own projects as you write them.

- click **File** then **Open Project...** to find it
- if you can't see the form then click the **View** menu and select **Form**

The form should look like this:



To run the program you can select **Start** from the **Run** menu - to stop it, click the Exit button.

If you run the program now you'll see various familiar features such as scroll bars, text boxes, check boxes and so on. Putting all this together to work as it does is far easier than you might think and Visual Basic provides you with all the tools that you need to start producing sophisticated and effective software with a minimum of effort.

This is really just a bit of fun but if you follow through the course you'll soon find yourself writing programs of one sort or another. Our advice is to experiment as much as you can - change things and try for yourself. Some things will work fine and others won't, but it's really the best way to learn.

Building Your First Project

It's traditional when learning to program - in virtually any programming language - to begin with a program called 'Hello', or something similar. This is just a very simple program which prints out some sort of message on the screen - usually "Hello, World!". That's what we'll do, and we'll use it to demonstrate some of the fundamental ideas behind Visual Basic. Here we go then...

- click **File** then **New Project**

The first thing we'll do is to save the form and save the project. They are saved separately because it may be that you want to use the same form for other projects and it saves you having to design the same thing over again.

- click **File** then **Save File As...**
- give the form a name, say, **hello.frm**
- click **File** then **Save Project As...**
- give the project a name, say, **hello.vbp**

OK so far? Remember to save your project whenever you make changes to it.

Now we're going to alter some of the properties for the form. There are over 40 properties listed and you could, of course, alter all of them. In practice, though, you'll soon get to know which are the important ones. Feel free to experiment with any or all of them. As far as we are concerned, we're going to leave most of them alone and we'll list the ones that we'd like you to check.

- to start with, we'd like you to check the values for the following properties on the form and alter them if necessary:

Form	**Name**	**frmHello**
	Caption	Hello
	Height	4000
	Left	2500
	Top	1600
	Width	6000

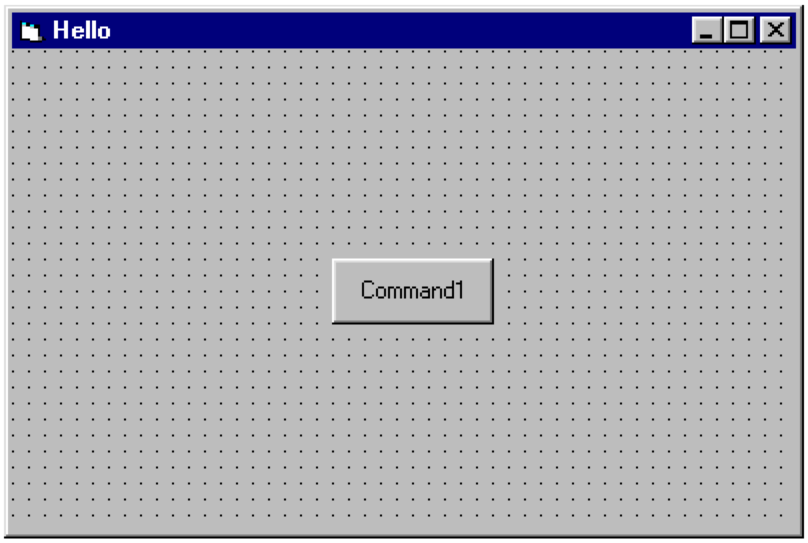
So that's decided the name, size and position of the form. Let's follow the convention of always naming a *form* with a name starting with **frm**. Note, by the way, that the Name property of the form is not the same as the filename that you choose to give the form when you saved it. If you want to set the background colour of the form then you can do so by clicking in the BackColor properties box, then clicking on the arrow that appears, and selecting a colour from the palette.

Now we're going to place a *command button* onto the form. There are two ways to do this:

1. double-click the command button icon in the toolbox - on our illustration earlier, it's the third one down in the right hand column

2. single-click the command button and then click and drag on the form to create a rectangle

Using the first method, you should have a form looking like this:



So now we're going to set some properties for the command button in a similar way to setting the properties for the form:

- click on the command button to highlight it
- note that the properties box now contains a list for the button
- check and adjust the properties for the button as follows

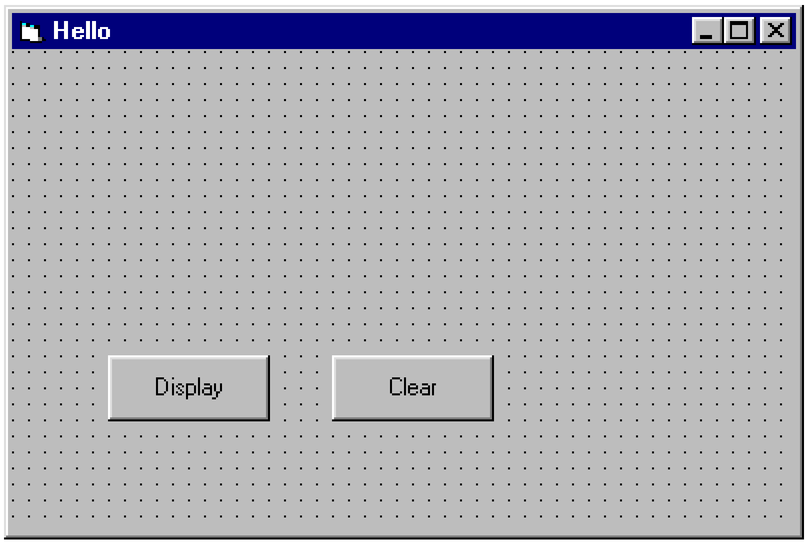
Command Button	**Name**	**cmdDisplay**
	Caption	Display
	Height	495
	Left	720
	Top	2280
	Width	1215

You'll notice when you make these changes, that the button now has a caption 'Display' and its position has been changed. We're going to follow the convention of always starting the name of a command button with **cmd**.

- add another command button in the same way and set the properties like this:

Command Button	**Name**	**cmdClear**
	Caption	Clear
	Height	495
	Left	2400
	Top	2280
	Width	1215

Your form should look like this now:

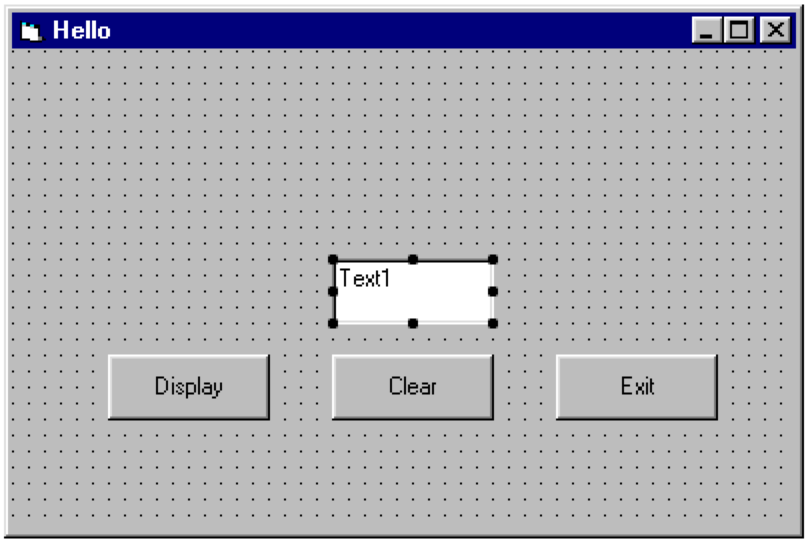


- add a third button with these properties

Command Button	**Name**	**cmdExit**
	Caption	Exit
	Height	495
	Left	4080
	Top	2280
	Width	1215

Now we're going to add another kind of control called a *text box*. In our illustration, it's the one above the command button in the toolbox.

- double-click the button to add the control - your form should look like this:



- you should set the text box properties as under:

Text Box	**Name**	**txtMessage**
	Alignment	Center
	Height	495
	Left	1680
	MultiLine	True
	Top	1320
	Width	2655

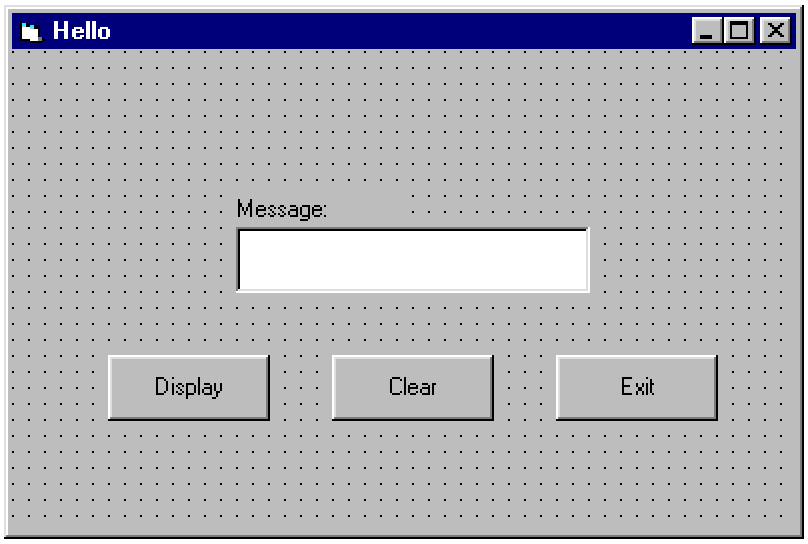
This time, we're going to adopt the convention of starting the names of text boxes with **txt**. It's a little quirky, but if you want to set the set alignment to Center, then we have to set the MultiLine property to True. To make sure that the box has no text in it, you need to click in the Text property box, click the arrow that appears, and then delete any text from the box which pops up.

Finally, we're going to add yet another type of control called a *label*. You'll find it second down from the top in the left hand column with a letter 'A' on it.

- double-click the button to add the label to your form
- set the properties as follows

Label	**Name**	**lblMessage**
	Caption	Message:
	Height	255
	Left	1680
	MultiLine	True
	Top	1080
	Width	1215

So your form now looks like this:



Make sure you save the project - clicking **File** then **Save Project** will automatically save the form as well as the project.

All well and good, but what's supposed to happen? To run the project look for the 'start' button on the toolbar - it's a small right-pointing black arrow.

- click the start button
- try clicking some of the buttons on the form

Nothing happens, but this isn't surprising really because all we've done is to design how the form *looks* and not how it *behaves*. If you like, think of this as the *visual* part of Visual Basic. Now we've got to look at the *programming* part of it...

The project is still running - this is indicated on the top title bar - and will be until you tell it to stop. To do this, you should now notice a button with a small black square on it just a little to the right of the start button - it was 'greyed out' before, making it inactive. After all, it doesn't make much sense to stop a project that hasn't been started does it? Likewise, when the project is running, the start button is similarly 'greyed out'.

- click the stop button

Before we start thinking about the programming code, let's just be clear about what we want the program to do. What is meant to happen when the user presses each of the buttons?

The button labelled Display should result in a message being displayed in the text box.

The button labelled Clear should result in any text in the text box being cleared.

The button labelled Exit should cause the program to stop running.

But why can't the user just press the Stop button on the toolbar? Well the short answer is that they can. What we're doing, though, is looking ahead to when we produce a self-contained *executable* program. Suppose, for example, that you wrote a games program that you thought was so good that other people might want to use it. You would need to create an executable version of your project so that other people could run it on their machines, bearing in mind that they might not have Visual Basic. Indeed why *should* they have Visual Basic if all they want to do is use your program? In other words, an *executable* program is one that can *execute* on their system without them needing to have Visual Basic installed.

It's an easy enough matter to create an executable file from your project and the option to do that is on the **File** menu.

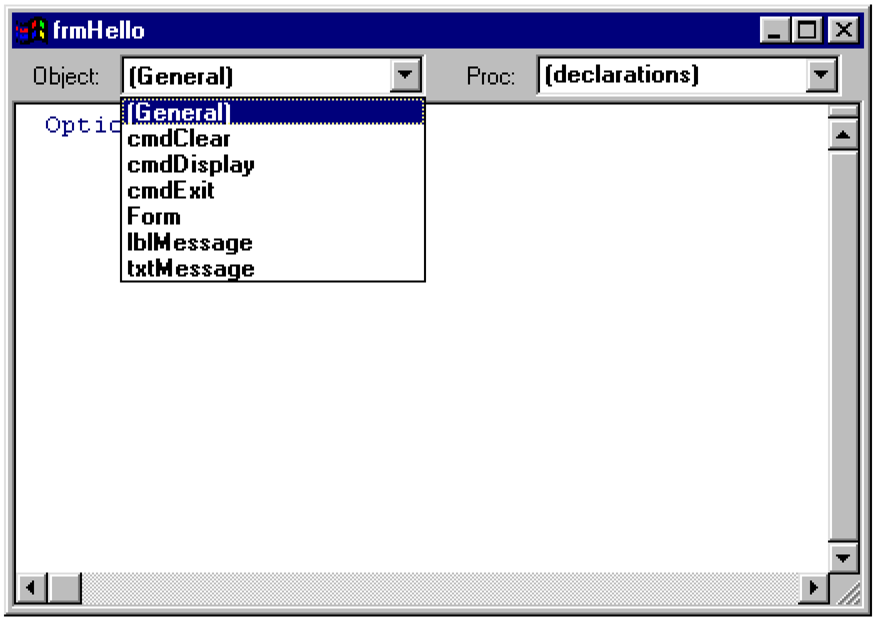
So, back to our project. The Exit button is the one that's going to stop the program and is the easiest to write the code for. We'll do that in just a moment but before that, we'd like you to check a couple of settings...

- click **Tools** then **Options...** and under the **Environment** tab make sure that **Require Variable Declaration** is ticked
- in the same dialog box, but under the **Editor** tab, make sure that **Full Module View** is *not* ticked

And now to the code...

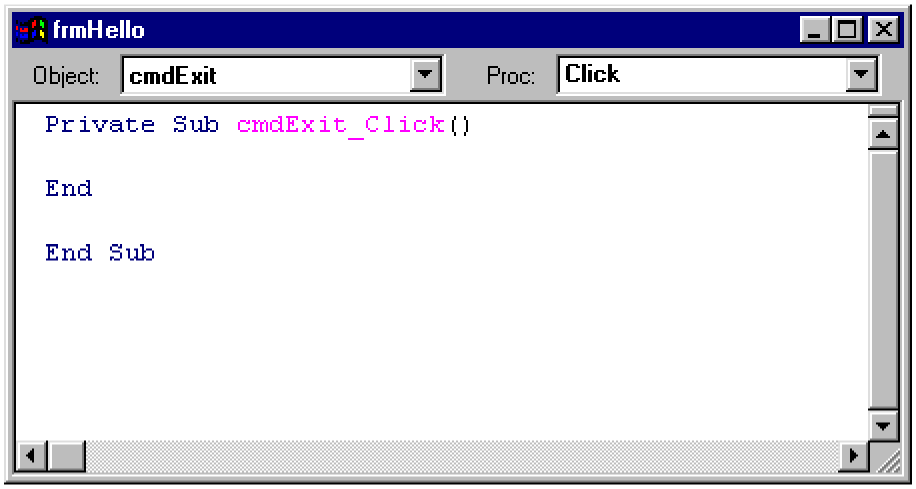
- in the Project window, click View Code

If you click the arrow in the left hand box at the top you should see something like this:



This is where the code for each of the controls is going to go. You don't have to write code for every control, only the ones where an *event* is going to occur. An event occurs, for example, when someone clicks a button or a checkbox, or when someone types text into a text box. In other words, an event is when the user *does* something, or *interacts*, with your program. Visual Basic is an *event-driven* language and your job as a programmer is to consider what events might occur and how you want your program to respond when they do.

- click on **cmdExit** and type in a line of text so that you have this:



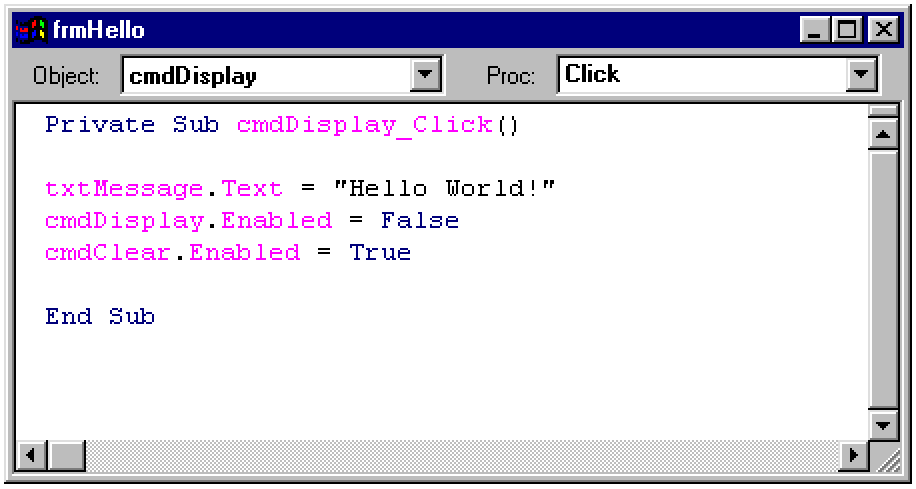
```
Private Sub cmdExit_Click()

End

End Sub
```

There now, didn't we tell you that programming was easy? This is the code for the Exit button. To be more specific, it's the code for the *Click event* of the Exit button. Notice that in the right hand box at the top it says **Click** and if you click the black arrow you'll see a list of other possible events associated with the **cmdExit** button.

- run the program and click the Exit button to check that it works

One of the properties of the **txtMessage** text box is called Text and, if you remember, we deleted any text there so that it was blank. It is possible to alter that property while the program is running and, indeed, that's exactly what we want to do when the user clicks the Display button. So we want the Click event of the Display button to alter the Text property of the text box control. Here's how it goes...

- click View Code and then select **cmdDisplay**
- type in the code to get this:



```
Private Sub cmdDisplay_Click()

txtMessage.Text = "Hello World!"
cmdDisplay.Enabled = False
cmdClear.Enabled = True

End Sub
```

The line of code

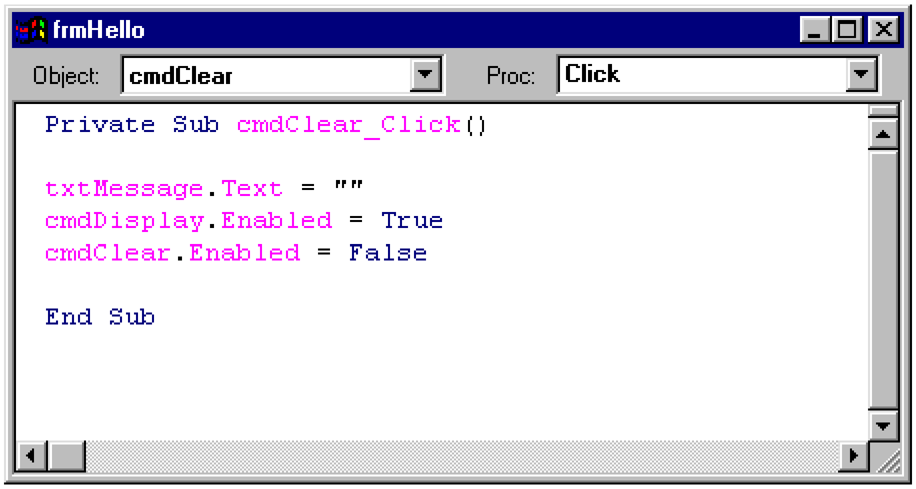
```
txtMessage.Text = "Hello World!"
```

is the one that sets the Text property of the **txtMessage** text box to "Hello World!". In other words, it displays your message.

- run the program and check that it works

So far, so good. We'll now add code to the Clear button whose job is to clear any text from the text box...

- add the code to the **cmdClear** Click event:



```
Private Sub cmdClear_Click()

txtMessage.Text = ""
cmdDisplay.Enabled = True
cmdClear.Enabled = False

End Sub
```

This time, the code

```
txtMessage.Text = ""
```

replaces whatever was there - your message - with whatever is between the quotes; in this case, nothing!

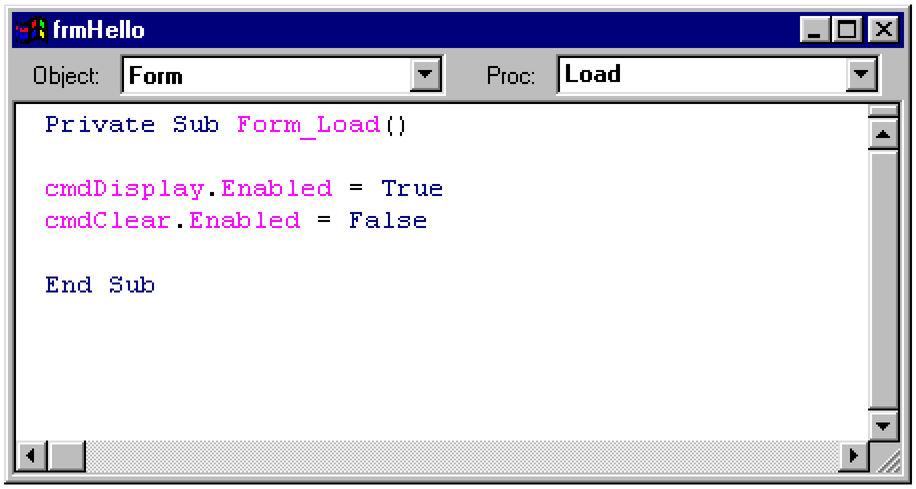
At this point, congratulations are in order because you've just written your first Visual Basic program!

Hello World Revisited

Recall that sometimes buttons are greyed out, or inactive, and that there are good reasons why this is desirable. In the Hello World example there's little point in having a button available which can Clear an already empty text box. Similarly, if the message is already being displayed, then we would prefer that the Display button is disabled.

We'll make few changes to the program...

- click View Code and select the **Form** control
- type in this code for the Load event:



```
Private Sub Form_Load()

cmdDisplay.Enabled = True
cmdClear.Enabled = False

End Sub
```

What this does is to *enable* the Display button and *disable* the Clear button when the form is first loaded. If you run this version of the program you'll see what happens....but you'll also discover that the program doesn't quite work as we would like it to.

- run the program and see if can discover what the problem is

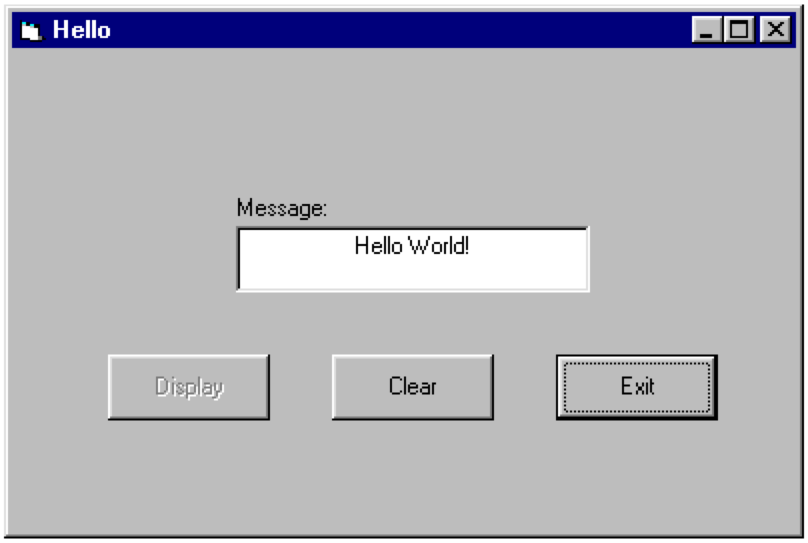
Here are some additions to the **cmdDisplay** control code...

```
Private Sub cmdDisplay_Click()

txtMessage.Text = "Hello World!"
cmdDisplay.Enabled = False
cmdClear.Enabled = True

End Sub
```

...and here are some more for the **cmdClear** control code:

```
Private Sub cmdClear_Click()

txtMessage.Text = ""
cmdDisplay.Enabled = True
cmdClear.Enabled = False

End Sub
```

- make these changes and run the program



You'll see that the program now works in a more acceptable way.

We can, in fact, go a step further and make the disabled button disappear altogether! As well as the Enabled property, there is also a Visible property which can also be set to True or False.

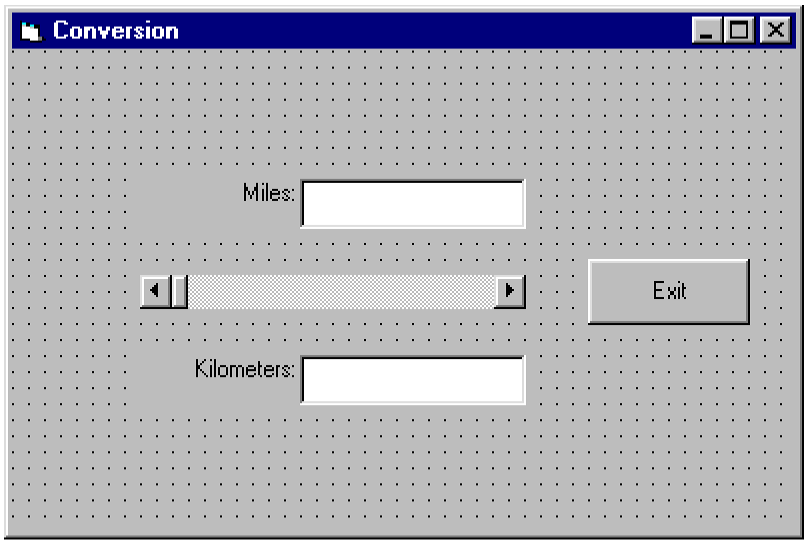
- see if you can make the necessary changes such that the Display and Clear buttons are only visible when they need to be

At this stage, you should appreciate that *forms* contain *controls.* The controls have various types of *properties* which can be set at *design time* and also at *run time*. The controls can also have associated *code* which is designed to deal with *events*. If you've managed to cope with all this, then you're over one of the biggest hurdles. If you're still a little puzzled, then let's work through another example and see if we can clear up any problems...

The Conversion Program

Scroll bars are useful controls and allow the user to enter information which can be used by the program. We're going to develop a program such that the user can specify a distance in miles which the program converts into kilometers. OK, it doesn't sound like it's going to win awards, but bear with us because it does bring out some important ideas...

The form is going to look like this:



- build the form according to this specification:
-

Form	**Name**	**frmConversion**
	Caption	Conversion
	Height	4000
	Left	2500
	Top	1600
	Width	6000
Command Button	**Name**	**cmdExit**
	Caption	Exit
	Height	495
	Left	4320
	Top	1560
	Width	1215

Text Box	**Name**	**txtMiles**
	Alignment	Center
	Height	375
	Left	2160
	MultiLine	True
	Top	960
	Width	1695
Text Box	**Name**	**txtKilometers**
	Alignment	Center
	Height	375
	Left	2160
	MultiLine	True
	Top	2280
	Width	1695
Label	**Name**	**lblMiles**
	Caption	Miles:
	Height	375
	Left	960
	Top	960
	Width	1215
Label	**Name**	**lblKilometers**
	Caption	Kilometers:
	Height	375
	Left	960
	Top	2280
	Width	1215
Horizontal Scroll Bar	**Name**	**hsbMiles**
	Height	255
	LargeChange	5
	Left	960
	Max	100
	Top	1680
	Width	2895

The last control - the horizontal scroll bar - is the sixth down in the left hand column of the toolbox in our illustration. Note the convention of giving it a name starting with **hsb**.

- run the program

You should check that there are three ways of moving the 'thumb' of the scroll bar. One is by clicking and/or holding the black arrows at either end of the bar. Another is by clicking and dragging the thumb itself. Thirdly, you can click in the bar to either side of the thumb to cause the thumb to move in larger 'jumps' - that's why we set the LargeChange property to 5.

- alter the LargeChange property to, say, 20 and see the effect

You'll see that, as well as moving in larger jumps, the width of the thumb has also increased.

The Max property of the bar was set to 100 and the Min property is 0 which gives us a range of values from 0 to 100 as the thumb moves along the bar. These are the values that we're going to use for the number of miles. Before we decide how to convert into kilometers, let's see if we can get the program to display the number of miles in the top text box.

If you've done any programming before, you'll probably have come across the idea of a *variable*. Our program is going to convert a number of *miles* into a number of *kilometers* and because we don't know in advance what these numbers are going to be, we need to use a couple of *variables*. One variable will be called `Miles` and the other will be called - you've guessed already - `Kilometers` and, as the name implies, they are going to *vary* depending on what the user chooses and what the program calculates.

What sort of numbers are they? Well `Miles` is going to be a whole number, or *integer*, between 0 and 100. `Kilometers`, on the other hand, will be calculated, and is probably going to have a decimal part to it. It's always worth considering these factors before trying to code the program.

The code for the **cmdExit** button is almost the same as for the last example but we've added just an extra bit of interest:

```
Private Sub cmdExit_Click()

Beep
End

End Sub
```

- enter this code and run the project

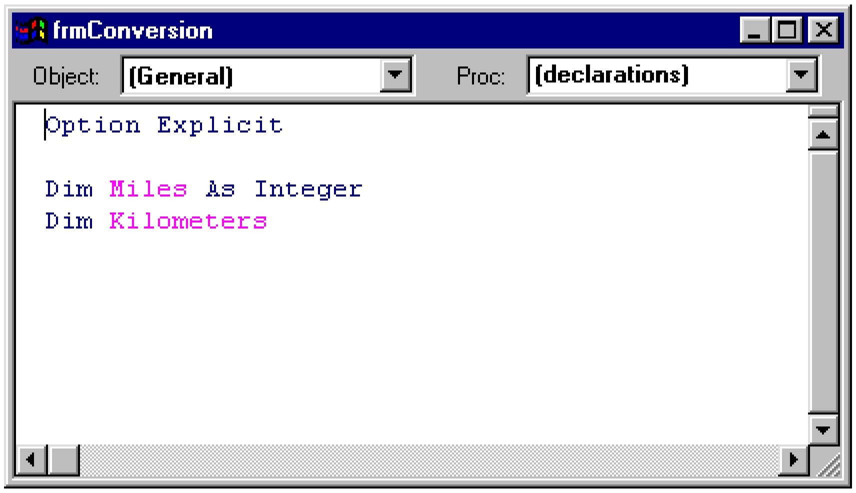
You should hear a beep through your computer's speakers when you exit the program. Some would find this irritating after a while but it can have it's uses at times. We just thought we'd mention it anyway...

You'll notice that one of the properties of the scroll bar is called Value. This is the key to the program and it's this that tells us the number of miles that the user has selected. We want our variable called `Miles` to take on this value and display it in the top text box.

To summarise, because it's important that you appreciate what's going on so far:

1. the user moves the scroll bar to select a value
2. the value is made available to the program by using the scroll bar's Value property
3. we're going to assign that value to a variable called `Miles`
4. we display the value of `Miles` in a text box

Firstly, we *declare* the two variables in the General Declarations section like this:



```
Option Explicit

Dim Miles As Integer
Dim Kilometers
```

- type in the code, remember to save your work

Some points to note here:

What we're doing is simply telling the program: "Hey, we're going to use a couple of variables called `Miles` and `Kilometers`. The first one, `Miles`, is definitely an integer; the second one, `Kilometers`, we're not too sure just now, so we'll leave it vague for the time being." You'll meet other types of variables later on in the course.

The phrase 'Option Explicit' appears because, earlier, we asked you to make sure that the Require Variable Declaration was ticked as one of the optional settings. What this does is to ensure that Visual Basic will complain if you try to use a variable without declaring it first. This is good practice, as you'll see later.

Turning to the code for the **hsbMiles** scroll bar, we have:

```
Private Sub hsbMiles_Change()

Miles = hsbMiles.Value
txtMiles.Text = Miles

End Sub
```

You'll notice that this is the Change event and so the code is executed whenever the user changes the Scroll Bar setting. This, of course, is exactly what we want and you can see that `Miles` takes on the value that the scroll bar is set to and then the Text property of the **txtMiles** text box, in turn, is assigned the value of `Miles`

- type in the code
- run the program

Well that seems OK so far. How about changing the number of miles into kilometers? You need to add a couple of lines of code as follows:

```
Private Sub hsbMiles_Change()

Miles = hsbMiles.Value
txtMiles.Text = Miles
Kilometers = 1.6 * Miles
txtKilometers.Text = Kilometers

End Sub
```

Like many other programming languages, Visual Basic uses the character '*' to mean 'multiply'.

- run the program now and it should do pretty well what we wanted it to

...pretty well, that is, but not *quite* all. Have you spotted the problem? If not, try this:

- run the program and click and drag the thumb of the scroll bar

The value is only displayed when you release the thumb. It would be much better if we could update the values as the bar is being scrolled. No problem. Look for the Scroll event in the **hsbMiles** section and add exactly the same code that appears for the Change event - copy and paste it to get this:

```
Private Sub hsbMiles_Scroll()

Miles = hsbMiles.Value
txtMiles.Text = Miles
Kilometers = 1.6 * Miles
txtKilometers.Text = Kilometers

End Sub
```

- run the program and check that it works as we intended

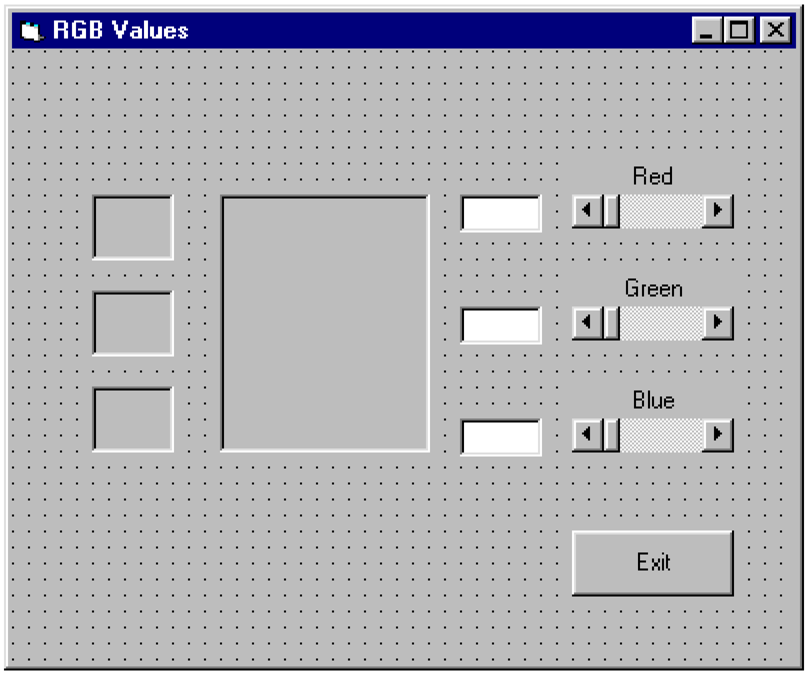
So now you've used *variables*. You know how to *declare* them and you've found a couple of ways to *assign* them a value. You've also learnt that sometimes you need to attach code for more than one event for the same control.

Traditionally, programmers have thought about programs in terms of *input, process* and *output*. In other words you get information, you do something with it and then you output the results to a screen or printer. This view has changed a little but is still relevant. The program you've just been using gets input from the scroll bar, processes it by calculating the number of kilometers and then outputs the results to a text box.

More on Scroll Bars

We'll develop a program here which demonstrates how colours work on your machine. Firstly, you need to be aware that the colours that you see have three components: red, green and blue. Each colour is given an *intensity* value - a number between 0 and 255 - and the colour that you see will depend on the mix of these three values.

The form that we're going to use will look like this:



- you need to build it according to the following specification

Form	**Name**	**frmRGB**
	Caption	RGB Values
	Height	5000
	Left	2500
	Top	1500
	Width	6000

Command Button	**Name**	**cmdExit**
	Caption	Exit
	Height	495
	Left	4200
	Top	3600
	Width	1215
Label	**Name**	**lblDisplay**
	Border Style	Fixed Single
	Caption	none
	Height	1935
	Left	1560
	Top	1080
	Width	1575
Label	**Name**	**lblBlue**
	Border Style	Fixed Single
	Caption	none
	Height	495
	Left	2500
	Top	2520
	Width	615
Label	**Name**	**lblGreen**
	Border Style	Fixed Single
	Caption	none
	Height	495
	Left	2500
	Top	1800
	Width	615
Label	**Name**	**lblRed**
	Border Style	Fixed Single
	Caption	none
	Height	495
	Left	600
	Top	1080
	Width	615
Label	**Name**	**lblRedScroll**
	Alignment	Center
	Caption	Red
	Height	255
	Left	4200
	Top	840
	Width	1215

Label	**Name**	**lblGreenScroll**
	Alignment	Center
	Caption	Green
	Height	255
	Left	4200
	Top	1680
	Width	1215
Label	**Name**	**lblBlueScroll**
	Alignment	Center
	Caption	Blue
	Height	255
	Left	4200
	Top	2520
	Width	1215
Horizontal Scroll Bar	**Name**	**hsbRed**
	Height	255
	Large Change	10
	Left	4200
	Max	255
	Top	1080
	Width	1215
Horizontal Scroll Bar	**Name**	**hsbGreen**
	Height	255
	Large Change	10
	Left	4200
	Max	255
	Top	1920
	Width	1215
Horizontal Scroll Bar	**Name**	**hsbBlue**
	Height	255
	Large Change	10
	Left	4200
	Max	255
	Top	2760
	Width	1215
Text Box	**Name**	**txtRedValue**
	Height	285
	Left	3360
	Text	none
	Top	1080
	Width	615

Text Box	**Name**	**txtGreenValue**
	Height	285
	Left	3360
	Text	none
	Top	1920
	Width	615
Text Box	**Name**	**txtBlueValue**
	Height	285
	Left	3360
	Text	none
	Top	2760
	Width	615

OK, well the program uses just three variables, one for each of the colour components. We'll call them, not surprisingly, `Red`, `Green` and `Blue`. They are, or course, integers and we will declare them in the General Declarations section:

```
Option Explicit

Dim Red As Integer
Dim Green As Integer
Dim Blue As Integer
```

Here's the code for the Change event of the Red scroll bar:

```
Private Sub hsbRed_Change()

Red = hsbRed.Value
txtRedValue.Text = Red
lblRed.BackColor = RGB(Red, 0, 0)
lblDisplay.BackColor = RGB(Red, Green, Blue)

End Sub
```

- enter the code and save your work
- run the program

When the user changes the value of the Red scroll bar, then, several things happen. Our variable, Red, takes on the value that the user sets - this happens in the first line of code:

```
Red = hsbRed.Value
```

Next, this value is assigned to the Text property of the text box using

```
txtRedValue.Text = Red
```

We want to use the small label called **lblRed** to show the red component so its BackColor property is assigned that value using

```
lblRed.BackColor = RGB(Red, 0, 0)
```

The *function* called RGB is part of Visual Basic and its job in life is to determine the value of the colour to be displayed. The green and blue components are both set to zero because we only want to show the red component here.

On the other hand, we must also update the larger label, **lblDisplay**, to show the overall colour which includes all three components. That is done using

```
lblDisplay.BackColor = RGB(Red, Green, Blue)
```

If you run the program, you'll come across the same problem as we had previously, inasmuch as when you hold and drag the thumb of the scroll bar, the display is only updated when you release it. The solution is straightforward, though, and involves copying the same four lines of code to the Scroll event of the scroll bar to get this

```
Private Sub hsbRed_Scroll()

Red = hsbRed.Value
txtRedValue.Text = Red
lblRed.BackColor = RGB(Red, 0, 0)
lblDisplay.BackColor = RGB(Red, Green, Blue)

End Sub
```

The Change events of the Green and Blue scroll bars are coded in a similar way:

```
Private Sub hsbGreen_Change()

Green = hsbGreen.Value
txtGreenValue.Text = Green
lblGreen.BackColor = RGB(0, Green, 0)
lblDisplay.BackColor = RGB(Red, Green, Blue)

End Sub
```

and

```
Private Sub hsbBlue_Change()

Blue = hsbBlue.Value
txtBlueValue.Text = Blue
lblBlue.BackColor = RGB(0, 0, Blue)
lblDisplay.BackColor = RGB(Red, Green, Blue)

End Sub
```

As before, you need to copy each of these to the corresponding Scroll events of the scroll bars.

- run the program and check that it works

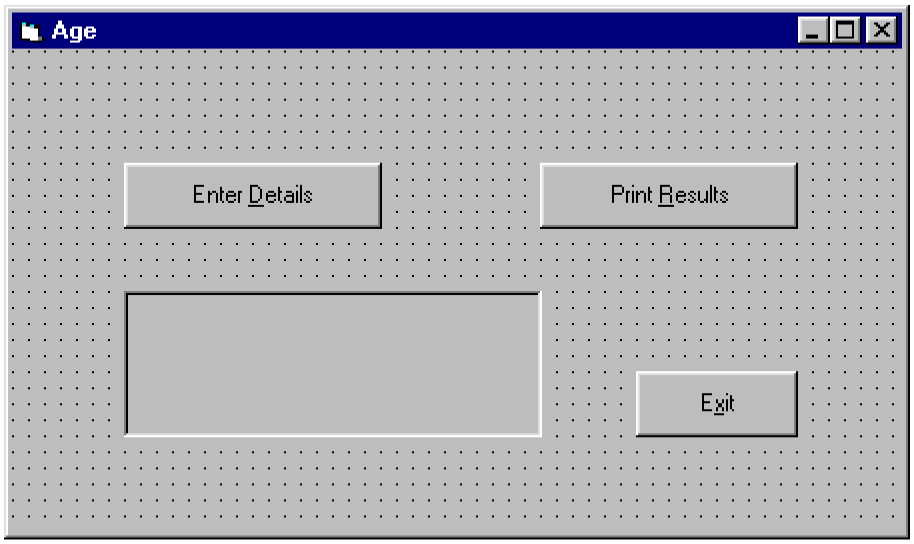
It's always nice to add a touch of colour - now you know how to do it!

Input Boxes

This program is going to use *input boxes* to allow the user to enter their name and age. It will do a simple calculation and print the results to a label.

We're also going to consider how to make our program more user friendly for those people who would rather use the keyboard than a mouse.

The form looks like this:



- build the form according to the following specification:

Form	**Name**	**frmAge**
	Caption	Age
	Height	4000
	Left	2100
	Top	1200
	Width	6800
Command Button	**Name**	**cmdExit**
	Caption	E&xit
	Height	495
	Left	4680
	Top	2400
	Width	1215

Command Button	**Name**	**cmdDetails**
	Caption	Enter &Details
	Height	495
	Left	840
	Top	840
	Width	1935
Command Button	**Name**	**cmdResults**
	Caption	Print &Results
	Height	495
	Left	3960
	Top	840
	Width	1935
Label	**Name**	**lblResults**
	Border Style	Fixed Single
	Caption	none
	Height	1095
	Left	840
	Top	1800
	Width	3135

Notice that the Caption property for each of the command buttons has an *ampersand* - a '&' character - in it and this has the effect of underlining the letter that follows it. When you run the program, users now have a choice of using the mouse to click on a button or using a key combination. Let's see how it works...

- enter the code for the Exit button in the usual way
- run the program
- hold the ALT key and press the 'x' key

So holding the ALT key and pressing the underlined letter has the same effect as clicking the button with a mouse. Clearly you must make sure that each button has a different letter that's underlined - you can't have two buttons with the same key combination. We're going to use this technique in all future projects.

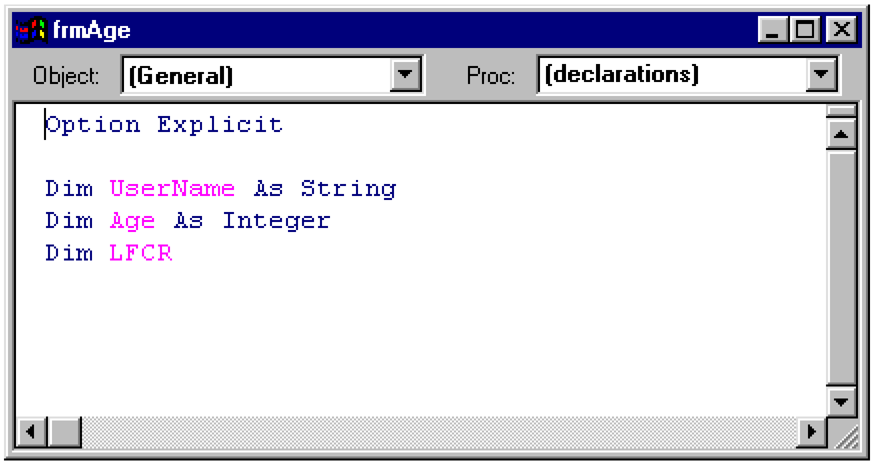
OK, so we want the user to type in their name and their age. It looks like we have two variables, then. The user's name is a *string*, or text, variable and we'll call it `UserName`. The person's age is an *integer* and we'll call it `Age`...

...and now for some history:

If you've ever used a typewriter - well, we did say *history* - you'll know that when you want to start a new line, you have to do a *line feed* and a *carriage return*. The first of these moves the paper up a line and the second moves you to the start of the line. Curiously, this has survived into the computer world, albeit in an electronic way, and we are going to need the electronic equivalent when we print out our results.

The upshot is that we really need to declare another variable called `LFCR` (Line Feed, Carriage Return) which we can use to do the job for us. You'll see how it works a little later but for now we need to declare all three variables in the General Declarations section of the project.

- declare the variables like this:



```
Option Explicit

Dim UserName As String
Dim Age As Integer
Dim LFCR
```

Here's the code for the **cmdDetails** Click event:

```
Private Sub cmdDetails_Click()

UserName = InputBox("Please enter your name:")
Age = InputBox("please enter your age:")

End Sub
```

- type in the code
- run the program

You see now what's meant by an input box. This is one of the joys of Visual Basic - you can use all these fancy tools with minimal effort...which leaves you free to concentrate on what your program should be doing.

Let's move now to the code for the Click event of the **cmdResults** button. This is a little more complicated...

```
Private Sub cmdResults_Click()

Dim Days As Integer
Dim TextOut As String

LFCR = Chr(10) + Chr(13)

Days = 365 * Age

'build output string
TextOut = "Hello " + UserName
TextOut = TextOut + LFCR
TextOut = TextOut + "You are at least "
TextOut = TextOut + Str(Days)
TextOut = TextOut + " days old!"

lblResults.Caption = TextOut

End Sub
```

We'll explain what all of this does in a moment.

- enter the code and save your work
- run the program and note what happens

OK, let's go through this code, starting with these two lines:

```
Dim Days As Integer
Dim TextOut As String
```

Nothing too mysterious here. We've declared `Days` as an integer and this is what we use for the person's age in days. We've also declared `TextOut` as a string and this is what we're going to use as the message to be printed out. But why didn't we declare them in the General Declarations section along with `UserName` and `Age`?

Well the answer is that we could have done and the program would still work. Notice, though, that `Days` and `TextOut` are only used in the **cmdResults** Click event and nowhere else. `UserName` and `Age`, on the other hand, are used in both the **cmdDetails** Click event and also the **cmdResults** Click event so it makes sense to declare them just once, right at the start. In general, it's good practice if a variable is only used in one *procedure,* or event, to just declare it where it's being used.

The line

```
LFCR = Chr(10) + Chr(13)
```

needs some explanation. `Chr(10)` is the linefeed character and `Chr(13)` is the carriage return character and so `LFCR` is a combination of the two, which we need when we print out the text in the label. OK, it's a bit messy but, put simply, all it does is move any following text onto a new line, just as if you'd pressed the ENTER key.

```
Days = 365 * Age
```

This calculates the number of days assuming that there are 365 days in a year - we don't want to worry about leap years here...

The next line

```
'build output string
```

doesn't in fact do anything and is ignored by the computer. It *does* however act as a *comment* to us and it is considered good practice to use comments in order to help other people understand your programs and make them more readable.

And now to the following code:

```
TextOut = "Hello " + UserName
TextOut = TextOut + LFCR
TextOut = TextOut + "You are at least "
TextOut = TextOut + Str(Days)
TextOut = TextOut + " days old!"
```

The comment said it all really. We are building up a string, or piece of text, step by step. We start with the word "Hello " and we add the variable `UserName`, which we know is another piece of text. The line

```
TextOut = TextOut + LFCR
```

takes the string `TextOut` and adds to it our linefeed/carriage return combination. We proceed to add to the `TextOut` string as we go. There is a slight problem because, while `UserName` is a string, `Days` is an integer, and it doesn't make sense to add a number to a string - they are completely different *data types*. The way round it, though, is to use a special function called `Str` which Visual Basic has thoughtfully provided. It's job is to take a numerical value and convert it into a string, which we can then add to `TextOut`. Easy, see?

- as an experiment, try removing the `Str` function and see what happens when you run the program using this line instead:

```
TextOut = TextOut + Days
```

You should get a 'type mismatch' error and if you click the 'Debug' button you'll find that the offending line of code has been highlighted.

Finally, the line

```
lblResults.Caption = TextOut
```

assigns our completed string to the Caption property of the label, and that's what you see printed out.

It's fair to say that we could have done all this using fewer lines of code - adding two, or more parts in a single line of code, for example - but we think it makes it clearer to set it out as shown.

Phew, heavy stuff, huh?

This program, then, has used different input and output methods to the previous one. It's very much a matter of preference and how you want your program to look. You should always consider how best to make your program easy to use. All we can do is to try and point out the possibilities. We're not quite done with this program yet, though, because we can use it to make a couple of important points...

Branching - If...Then...Else...End If

One of the most fundamental features of a computer is its ability to make decisions. By that, we don't mean intelligent decisions, but simple ones, depending on whether something is true of false. This lies at the heart of virtually all programs and it's something we need to look into.

Let's illustrate what we mean by adding some code to the Age program which we've just been looking at.

- edit the Click procedure of the Results button to look like this:

```
Private Sub cmdResults_Click()

Dim Days As Integer
Dim TextOut As String

LFCR = Chr(10) + Chr(13)

Days = 365 * Age

'build output string
TextOut = "Hello " + UserName
TextOut = TextOut + LFCR
TextOut = TextOut + "You are at least "
TextOut = TextOut + Str(Days)
TextOut = TextOut + " days old!"
TextOut = TextOut + LFCR

If Days > 15000 Then
  TextOut = TextOut + "You look not a day older than 10000!"
Else
  TextOut = TextOut + "Ah, just a youngster, then!"
End If

lblResults.Caption = TextOut

End Sub
```

- run the program choosing some suitable numbers to test it

So what the program prints depends on the age of the user...or, at least, on the number that they type in - the program isn't *that* smart! This is so important, we're going to ask you to look at it again. The general form of this extra bit of code looks something like this:

```
If (condition) Then
  (do something)
Else
  (do something else)
End If
```

...and when it says

```
If (condition)
```

it really means "If the condition is True..."

We will also note that the "`Else`" part is optional, so we could have something like:

```
If (condition) Then
  (do something)
End If
```

The "`End If`" part, though, is not optional and must be there, otherwise you will get an error message.

Notice that what follows the "`If`" statement is *indented*, that is, set in a little from the left margin. This is good practice and makes your programs easier to read. Anything following the "`Else`" statement is similarly indented until we reach the "`End If`" statement. Try to get into the habit of doing this.

We will also take this opportunity to point out that if you ever need to look for help, the Help index that comes with Visual Basic is very comprehensive and also has examples to show how various parts of code are used.

In the program that you've just been looking at, for example, we used an input box. If you look at input boxes in the Help index you'll see that there are ways in which you can control how and where the box appears, what should go in its title bar, and so on. There isn't space to go into that level of detail here and, in any case, it's probably best left for you to explore at your leisure - but be aware that you have that option.

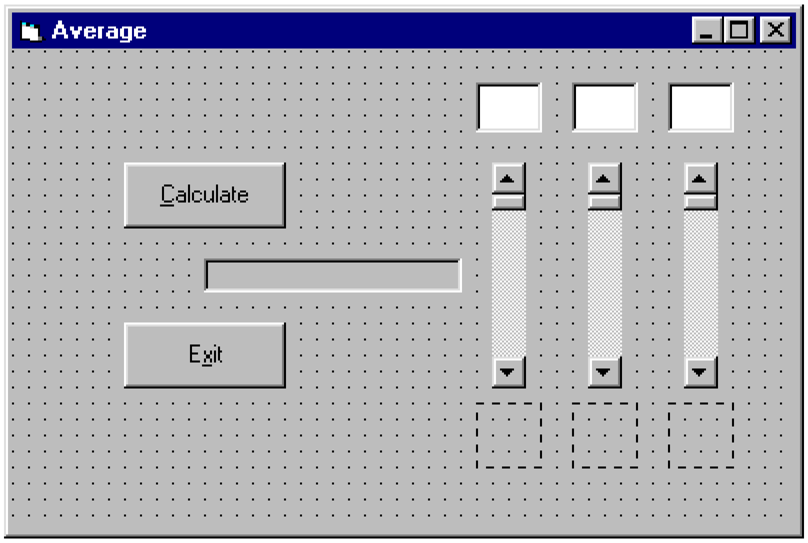
On a similar note, and when you're working through any of these programs, you are encouraged to experiment with different designs for the forms. Ours are just very basic designs with no frills, but there's no reason why you can't set different properties for the various controls - size, position, colour, text, borders and so on.

Don't, however, get carried away down this road and keep a focus on the main ideas that we're trying to get across...

Averages

This next example will calculate the average mark of three students who sit an exam. They will pass or fail according to the quite brutal principle that you pass if you score better than the average mark...

Here's our design for the form:



The specification is as follows:

Form	**Name**	**frmAverage**
	Caption	Average
	Height	4000
	Left	2000
	Top	2000
	Width	6000
Command Button	**Name**	**cmdExit**
	Caption	E&xit
	Height	495
	Left	840
	Top	2040
	Width	1215

Command Button	**Name**	**cmdCalculate**
	Caption	&Calculate
	Height	495
	Left	840
	Top	840
	Width	1215
Image	**Name**	**imgA**
	Height	495
	Left	3480
	Stretch	True
	Top	2640
	Width	495
Image	**Name**	**imgB**
	Height	495
	Left	4200
	Stretch	True
	Top	2640
	Width	495
Image	**Name**	**imgC**
	Height	495
	Left	4920
	Stretch	True
	Top	2640
	Width	495
Label	**Name**	**lblAverage**
	Border Style	Fixed Single
	Height	255
	Left	1440
	Top	1560
	Width	1935
Text Box	**Name**	**txtA**
	Alignment	Center
	Height	375
	Left	3480
	MultiLine	True
	Top	240
	Width	495

Text Box	**Name**	**txtB**
	Alignment	Center
	Height	375
	Left	4200
	MultiLine	True
	Top	240
	Width	495
Text Box	**Name**	**txtC**
	Alignment	Center
	Height	375
	Left	4920
	MultiLine	True
	Top	240
	Width	495
Vertical Scroll Bar	**Name**	**vsbA**
	Height	1695
	Large Change	10
	Left	3600
	Max	100
	Top	840
	Width	255
Vertical Scroll Bar	**Name**	**vsbB**
	Height	1695
	Large Change	10
	Left	4320
	Max	100
	Top	840
	Width	255
Vertical Scroll Bar	**Name**	**vsbC**
	Height	1695
	Large Change	10
	Left	5040
	Max	100
	Top	840
	Width	255

- build the form and save your work

It seems clear that we will need three variables - one for each of the marks - and another one for the average. We'll call them MarkA, MarkB, MarkC and Average. They are declared in the General Declarations section:

```
Option Explicit

Dim MarkA As Integer
Dim MarkB As Integer
Dim MarkC As Integer
Dim Average
```

We note that MarkA, MarkB and MarkC are integers but that Average is unlikely to be an integer so we just declare it but don't specify a type.

- enter the code
- enter the code for the Exit button

Consider, then, exactly what we want the program to do. The idea is that the user will set the three exam marks using the scroll bars. When the user clicks the Calculate button, the program will take the three marks, calculate the average and display it in the label. It will also display images in the image boxes - a 'smiley' if the person has passed, and a 'not-so-smiley' if they have failed...

So where do these images come from? Well Visual Basic comes with a selection of icons and bitmapped images. Have a look on your system for a sub-directory wherever Visual Basic has been installed - something similar to:

c:\vb\icons\misc\

and the two images we're looking for are called **face01.ico** and **face02.ico**. If you can't find them on your system then you could draw your own using a paint package or you could use any other images with a **.bmp** or **.ico** file extension. The images themselves are not important, it's how they are used that you should be focusing on.

Let's look at some of the code then.

- enter the familiar code for the Exit button
- enter the following code for the Change event of the leftmost vertical scroll bar:

```
Private Sub vsbA_Change()

txtA.Text = vsbA.Value

End Sub
```

- copy the same line of code to the Scroll event of the scroll bar
- you should now be able to enter code for the Change and Scroll events of the other two scroll bars in a similar way
- run the program to check that things are working OK so far
- save your project

Now we come to the code for the Calculate button and this is where most of the action takes place. It looks like this:

```
Private Sub cmdCalculate_Click()

'assign marks to variables
MarkA = vsbA.Value
MarkB = vsbB.Value
MarkC = vsbC.Value

'calculate average
Average = (MarkA + MarkB + MarkC) / 3

'display the average mark
lblAverage.Caption = "average is " + Str(Average)

'display the pictures
If MarkA > Average Then
  imgA.Picture = LoadPicture("c:\vb\icons\misc\face02.ico")
Else
  imgA.Picture = LoadPicture("c:\vb\icons\misc\face01.ico")
End If

If MarkB > Average Then
  imgB.Picture = LoadPicture("c:\vb\icons\misc\face02.ico")
Else
  imgB.Picture = LoadPicture("c:\vb\icons\misc\face01.ico")
End If
```

```
If MarkC > Average Then
  imgC.Picture = LoadPicture("c:\vb\icons\misc\face02.ico")
Else
  imgC.Picture = LoadPicture("c:\vb\icons\misc\face01.ico")
End If

End Sub
```

You'll see that this is really in four sections and we've put a comment line for each one. The first part should be familiar to you and simply assigns the values that the user has set to each of the three variables `MarkA`, `MarkB` and `MarkC`.

The second part

```
Average = (MarkA + MarkB + MarkC) / 3
```

calculates the average. Again like most languages, Visual Basic uses a '/' character to mean 'divide'. The '>' character means 'is greater than'.

To print the average value on the label, we use

```
lblAverage.Caption = "average is " + Str(Average)
```

which assigns it to the Caption property. Note that we must use the `Str` function again to convert the *numerical* value into a *string* value and note also that the answer is printed to five decimal places when necessary. We'll return to the topic of *formatting* the output at a later time.

The section of code

```
If MarkA > Average Then
  imgA.Picture = LoadPicture("c:\vb\icons\misc\face02.ico")
Else
  imgA.Picture = LoadPicture("c:\vb\icons\misc\face01.ico")
End If
```

checks to see if the mark is greater than the average. If it is, then we use another function, called `LoadPicture` to assign the specified picture file to the Picture property of the image control. If it isn't, we assign the other picture instead. In order to clear the image box altogether, you could use a line of code like

```
imgA.Picture = LoadPicture("")
```

You will also note that we set the Stretch property of the image controls to True.

- set the Stretch property to False and see what happens
- try resizing the image controls and note the effect that the Stretch property has

Now we did say that this system is brutal. If the three marks are, say, 40, 40 and 40 then *nobody* passes. Why is this? The answer lies in the way we decided who is to pass. If the three marks are 40, 40 and 40 then the average is also 40 and, of course, no one has got a mark which is actually *greater* than 40. If what we meant was *greater than or equal to* 40 then the code would be slightly different and you would use something like

```
If MarkA >= Average Then
```

- make the three changes necessary and run this new version

Hey, this is *much* better, now *everyone* can pass!

So now you know how to use images in your programs and you can alter them at run time. We'll note, in passing, that if you wanted other people to be able to use this program on their machines, then it isn't necessary for them to have copies of the images installed. When you make an *execuatble* copy of your program, any images are incorporated in it automatically.

ePublish Scotland

Evaluation Copy

This is the evaluation copy of the Visual Basic tutorial and represents about a quarter of the full version.

We hope that you've found it useful - if it has inspired you to go on to greater things then that's a bonus and we're delighted.

The complete version of the tutorial runs to around 190 pages and is distributed on the shareware principle for the modest sum of 12 UK Pounds (around US$ 20). It is also available in HTML format and there are two versions - one covering Visual Basic version 4 and the other covering Visual Basic versions 5 and 6.

Further details of this, and other, products can be found on our website where you'll also find links to a wealth of other educational sites - enjoy!

Your comments and feedback are welcome.

email at: info@epublish-scotland.com

website at: www.epublish-scotland.com