

Chapter 2

Controls used in Visual Programming

Chapter summary

This chapter opens with the development environment for constructing Visual Programs, and the code which makes them up. Event-driven systems and the ways in which they may be executed are introduced. Some of the components used are described in more detail, and a simple event-driven Visual Program is constructed.

Learning outcomes

The learning outcomes for this chapter are associated with the development of a simple Visual Program. When you have completed the chapter successfully, you will be able to:

- Outcome 1:** Describe the environment in which Visual Programs are constructed,
- Outcome 2:** Describe some of the components which are used in Visual Programs, and their properties.
- Outcome 3:** Construct a simple Visual Program.
- Outcome 4:** Discuss simple basic event-driven programming concepts.
- Outcome 5:** Use ways in which further information can be discovered.
- Outcome 6:** Describe some Software Engineering principles which apply to Visual Programming.
- Outcome 7:** Customise the appearance of your Visual Program.

How will you be assessed on this?

You will construct a simple Visual Program. You will be asked questions about the tools and controls used to construct a viable user interface. Some questions will have numerical answers.

Section 1

Building a Visual Program

In this section, you will learn about the environment in which Visual Programs are developed.

To become familiar with Visual Programming – in this case Visual Basic – it will be useful to discover the environment in which it works. So that you can be active in your learning, start by launching the system (as mentioned in Chapter 1, the descriptions and illustrations come from Visual Basic 6.0). You are offered a set of initial choices, shown in Figure 2.1. As this is your first program, open the new Standard EXE option under the *New* tab. (When you access an existing program for maintenance, you will use the *Existing* tab.)



Figure 2.1 Initial choices

The Visual Basic system has opened in Design Mode, and is showing windows for a Project and a Form. The form is the visible part of the Project. (If it is not showing, select *Object* from the *View* menu, or use *Shift-F7*.)

CRUCIAL CONCEPT

Visual Basic is flexible: there is usually more than one way of carrying out any activity in Visual Basic.

Other windows should also be showing: in particular, the Toolbox. If this is not visible (it will look something like Figure 2.2), select *Toolbox* from the *View* menu, or use the **Toolbox** button in the upper toolbar (shown on the right of Figure 2.2). You will need the Toolbox when constructing a Visual Program. It allows you to construct the **controls** described below.



Figure 2.2 Toolbox and Toolbox button

CRUCIAL ACTIVITY

Launch Visual Basic and open the initial Design Mode Project with its form. Make sure the Toolbox is showing.

Quick test

1. Name two ways of displaying the current form in Visual Basic.
2. How many tools are shown in the Toolbox?

Section 2

Visual Programming forms and code

In this section, you will learn about the main component of a Visual Program, the units used for measuring the screens produced, and the way in which program code is stored and displayed.

As an actual Visual Program is being executed, there is a 'Master Component' which is intended to contain all of the others. This allows the components to be placed at picked locations in a controlled way. This component is the **form**, mentioned in Section 1. It is an important part of any Visual Program, and it is possible to produce such a program with no other component. This is not a realistic goal, however: it corresponds to using a computer screen as a single input/output mechanism, with no additional structure. However, to demonstrate the capabilities of a form and the use of some of the associated programming tools, it will be used to show some simple output.

CRUCIAL CONCEPT

The **form** is used in Visual Basic to contain other components.

The appearance of the form in Design Mode shows its actual size, but not necessarily where it will appear when the program is running. The size is also shown on the toolbar, as is the position. The default measuring units used by Visual Basic are called **twips**, although this can be changed. A twip is 1/20 of a printer's point ('**tw**entieth of a **p**oint'), of which 72 make up 1 inch. Thus there are 1,440 twips per inch, and roughly 567 twips per centimetre.

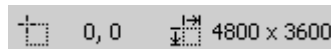


Figure 2.3 Position and size of form

In Figure 2.3 the top left corner of the form is shown to be at coordinates (0, 0), and the form has a width and height of 4,800 and 3,600 respectively. (That shows the form to be roughly 8½ cm by 6¼ cm in size.) The width and height may be changed by dragging the right side or the bottom of the form. The size display will change accordingly. This will be further explored in Section 7 below.

CRUCIAL CONCEPT

Measurement in Visual Basic is based on the **twip**, which is 1/1440 inch.

Program code is associated with each component of a Visual Program. The active parts are organised into a set of subroutines, which are formally known as **methods**. These methods may be viewed by selecting *Code* from the *View* menu. This opens a window with three components: two small ones above, and a larger one below. The smaller windows, which are known as **combo boxes**, will contain (*General*) on the left, and (*Declarations*) on the right. In the left-hand box, using the arrow, select *Form*. The right-hand box will change to *Load*, and the text

```
Private Sub Form_Load()  
End Sub
```

will appear in the larger window component. This is one of the methods which has been prepared in advance for the Visual Programmer. In this case, it is the code to be executed

when the form is first set up during the running of the program. There is also a short cut to this code window, by giving a double click in the form in its design view. This is sometimes a more convenient way to access the code. This method will be seen again in Chapter 8, used for initialisation purposes.

CRUCIAL CONCEPT

All Visual Basic is based on **program code** stored as text. The code is organised as a set of **methods**.

The code statements to be executed at this time are inserted between the two lines. These may be (more or less) any valid Visual Basic statements. However, it is reasonable to expect that these would be prepared according to appropriate principles of design, and laid out accordingly. An example of possible code statements to be included in a program is shown below. It will allow for simple output, but will also be used to give an example of parts of a Visual Program which may be included with no extra work from the programmer.

CRUCIAL ACTIVITY

Open the code window and check its contents as described above.

Quick test

1. What does twip stand for? What is the size of a twip in inches? In centimetres?
2. How is program code organised in Visual Basic?

Section 3

A simple event-driven program

In this section, you will be introduced to the organisation and some of the names of a Visual Program's methods. You will also learn how to start and stop the execution of a Visual Program.

Starting from the open code window, the right-hand combo box will show a list of methods which apply to forms. The list is quite a long one, and to see all of the method names a slider is provided to access the entire list. The names are generally self-explanatory: for example, *KeyPress* and *MouseMove*. These methods are called whenever a specific **event** happens which is associated with the form. In the example, the first routine is entered whenever a key (on the keyboard) is pressed; the second is invoked whenever the mouse is moved.

CRUCIAL CONCEPT

Methods are executed in response to **events**.

When the system is initialised, each of these methods is present only as an outline. If any one is selected, its outline code appears in the large window, with the appropriate parameter list in each case. If no code is added, then there is nothing to execute if the given event occurs. It is within these methods, therefore, that code is to be added. For the purpose of this example, simple text will be output whenever the mouse button is used.

This operation is known to the Visual Basic system as a **click**; the associated method for the **form** has the name *Form_Click*. It takes no parameters, as the empty parameter list shows when the method is selected.

CRUCIAL ACTIVITY

Select the **Click** method contained in the form's code. Check the layout of the skeleton code.

The program to be written will consist of a single line of active code, and is inserted after the subroutine heading. However, to ensure good initial programming practice, an explanatory comment will also be added. This will come first, and is introduced, as in many other examples of the Basic language, with an apostrophe. After this, any text can be written on a single line, and will be ignored by the system. (If you prefer, the keyword *Rem* will also introduce a comment.) To distinguish comments, they are given distinctive colours: the default is green. This helps to pick them out from the code statements which are to be executed.

The statement to produce output starts, as in other Basic programs, the keyword *Print*. The text to be printed is then given as a string of characters, placed within quotation marks. This may be any text at all; a simple message has been selected as an example. When this has been prepared, the program is ready to execute.

```
Private Sub Form_Click()
    ` Output a line each time the mouse button is pressed
    Print "Test message"
End Sub
```

In the same way that comments are presented in a distinctive colour, so are the keywords which are recognised by the Visual Basic system. In the example, the words *Private*, *Sub*, *Print* and *End* all appear with a default colour of dark blue: again, for ease of recognition during program development.

To execute the program, there are three equivalent ways. The command *Start* may be used from the *Run* menu; or the **run** button on the toolbar may be pressed; or the *F5* key may be used. In each case, the form appears in the position specified (see Figure 2.3). However, nothing else happens, because no significant event has taken place. To activate the method, the program must recognise a mouse click associated with the form. As soon as this has been done, the line of text is printed; and this is repeated each time the event takes place.

To stop the program, the command *End* may be used from the *Run* menu; or the **stop** button on the toolbar may be pressed; or the form window may be closed using its associated **close** button in the top right-hand corner. In each case, program execution stops and the form window is closed.

CRUCIAL ACTIVITY

Insert appropriate comments and commands into the skeleton subroutine code. Execute the program.

Quick test

1. Identify five different methods associated with a form.
2. List two ways to start a Visual Basic program.

Section 4

Visual Programming controls

In this section, you will be introduced to different types of Visual Basic components and how they are included in a Visual Program.

In addition to the form, there are several components used in Visual Programming. These components are called **controls**. They should be familiar to users of programs based on

windows, and fall into two main groups. Some are intended for the users – **active** controls. Examples of there are **buttons** of different kinds and **text boxes**. These are shown in Figure 2.4.

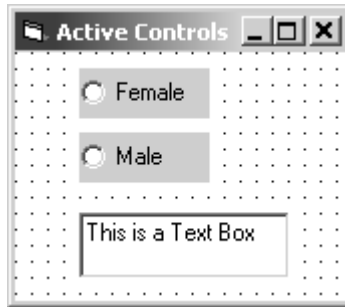


Figure 2.3 Examples of active controls

Others are meant for use by the programs you write, perhaps to give information to users. These are **passive** controls: for instance, pictures and labels. Examples of these are shown in Figure 2.5.

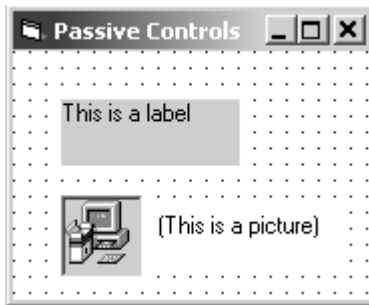


Figure 2.5 Examples of passive controls

CRUCIAL CONCEPT

Visual Programming relies on its **controls**. There are many of these. Some examples are buttons, labels and text boxes.

Active controls are intended for users to interact with. Passive controls are to give information to users. Each kind is set up on the form in the same way, using the toolbox.

The control to be used is selected from the toolbox, using the corresponding button. It is then positioned on the form as a 'drag and drop'. This allows the control's position and size to be specified at the same time. The form is provided with points spaced on a regular grid, to assist with lining controls up in an acceptable pattern. If the control has been positioned or sized incorrectly, it may be moved or resized in the standard way.

CRUCIAL ACTIVITY

Select appropriate controls and place them on the form. If the form becomes too small to hold them, resize it. Adjust the positions and sizes of the controls you have placed on the form.

There are several types of control available to the programmer: 20 are shown in the toolbox in Figure 2.2. It is not proposed to describe them all in detail at this point. They will be introduced, as they are needed, in following sections. Their properties and uses may be found by using the help system, as described in the next section. There appear to be 21

controls, but the arrow shown in the upper left of Figure 2.2 does not actually define a control. It is an arrow tool to help in grouping items. This is dealt with later in the text.

Quick test

1. Name five different Visual Basic controls.
2. How are Visual Basic controls included in a Visual Program?

Section 5

Use of Visual Basic Help

In this section, you will be introduced to the Help facility included with Visual Basic.

There are many more aspects of Visual Programming, as demonstrated using Visual Basic, than can be shown in this book. It is one of your tasks to use the Visual Basic system to identify these additional aspects, and to learn about them.

You will do this by using the Visual Basic Help facility. This can be started from the Help label on the menu bar. You can do this at any time. The result is shown in Figure 2.6.

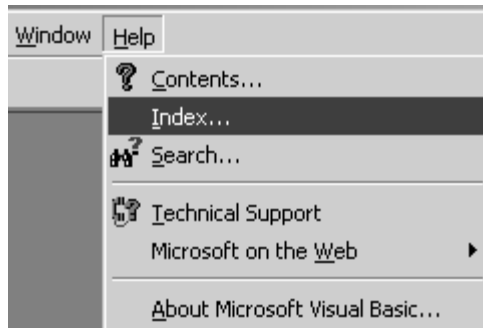


Figure 2.6 Help screen facility

CRUCIAL CONCEPT

Visual Basic provides a powerful **Help** facility.

You can see that the initial Help drop-down menu has a number of options. Select the Index tab and type a subject in the keyword area. This sequence is shown in Figure 2.7, using the keyword 'logical'.

You can also do this directly by using the *F1* key. There are additional facilities made available in this way, as you will find out later.

CRUCIAL ACTIVITY

Open the Help index, and type 'button' (but without the quotation marks) – or any other word that should apply to Visual Programming. Make a note of what you find out, and how you did it.

CRUCIAL TIP

You can find out about (nearly) anything in Visual Basic at (almost) any time by using the Help facility.

When you have finished looking anything up, you can minimise (or close) the Help screen in the usual way. You would probably wish to close it if you are sure that you have learnt what you need to know. You might wish to minimise it if it contains a series of steps you need to follow. You can always call it back again when you need it.

If you find something of particular importance, you might want to print the contents of the Help item. You do this most easily using the Print icon (circled in Figure 2.7).

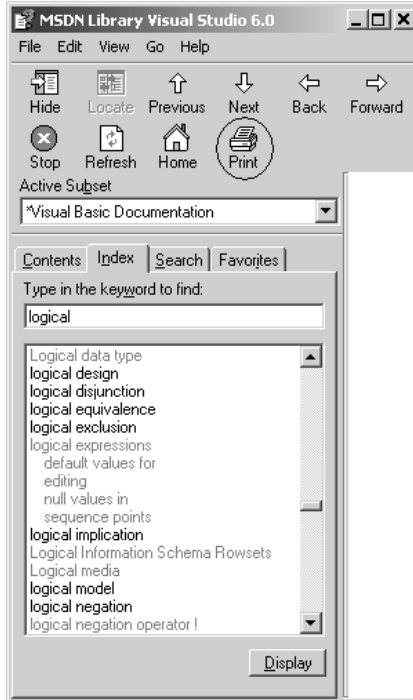


Figure 2.7 Help screen in use

CRUCIAL TIP

When you print anything, make sure you keep it safely and well organised. You might also wish to add other (handwritten) notes of your own, such as **what** you were doing at the time, **why** you printed them, and **when** you did it. Your author's notes have headings, dates, and times and sequence numbers on them. They are also filed in a binder.

Never worry about asking for help: it is a good way to start learning. Rudyard Kipling put it very well in *Just So Stories*:

'I keep six honest serving-men
 (They taught me all I knew);
 Their names are What and Why and When
 And How and Where and Who.'

Quick test

1. Name two ways of accessing Visual Basic Help.
2. How do you make a printed record of information from the Help system?

Section 6

Visual Programming elements

In this section, you will be introduced to Visual Basic projects and the parts which make them up.

When you are constructing a typical Visual Program, it will consist of a number of different parts. You can develop the parts separately, in a modular fashion. This is usually a good idea anyway. If you develop your work in this way, you can reuse common parts across different applications. This means – apart from less work – that common elements need to be designed, implemented and tested only once.

CRUCIAL CONCEPT

The reuse of common tried-and-tested programming components is one of the fundamentals of **Software Engineering**.

But then, how does a given Visual Program know which parts it is to use? This is done by means of a **project**. This is represented by a file with extension **.MAK**. Each such file defines the contents of the corresponding program. (For the reasons why this extension is used, you might like to look up the (UNIX) program **make**.)

These items have been carefully organised in a way which makes the most of its host computer's power. For instance, you can execute an existing Visual Program without making any alterations to it. If you then close the Visual Basic system, it will do so without any fuss. But if you make changes and then try to close it down, you will be asked if you want to save those changes (of course, you don't have to if you don't want to). This is part of the work of the project (.MAK) file.

During the next few exercises you will find different project components. Each of these has its particular place inside a project. Therefore, each component will be recorded in the project file. Each different kind is itself held in a file, with a particular extension according to its organisation. If a file has been changed, you will be invited to save that file as the project is closed. This helps to improve the general efficiency of the work being done. No change – no extra work.

To find the contents of a project, look in the project window. This is found as Project Explorer on the View menu, or from ctrl-R on the keyboard. It shows the major project components arranged as a tree. In this way, the dependence of each item on the rest of the structure is readily visible.

Quick test

1. How are the contents of a project displayed?
2. How does Visual Basic use its storage time efficiently?

Section 7

Control properties

In this section, you will be introduced to some of the properties of Visual Basic controls.

There are **properties** associated with each control, and a form also has its own. These define its size, shape, position, colour and many other attributes. Those showing the size and position of the current form have been partially described in section 2 above. These will now be looked at in further detail.

The characteristics associated with the currently active control may be found in the properties window. This is activated by the key *F4*, or by selecting the *Properties Window* from the *View* menu.

There are three main parts in the window. These are the control to which it refers, the properties of that control, and a description of the property which is currently selected. In the example shown in Figure 2.8, the component is a form whose name is *Form1*. This can be seen in the combo box at the top of the figure.

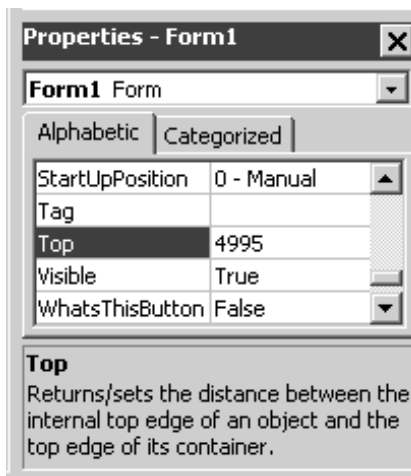


Figure 2.8 Control properties

The property selected in this example is *Top*. This defines where the top of the form appears when the program is running. This is shown in Figure 2.8 at position 4995, or about 9 centimetres from the top of the main window. More information about the selected property is shown in the bottom part of the window. The information in the example is a description of the *Top* property.

Some of the properties are set at the time that a Visual Program is designed and constructed, and cannot afterwards be altered – without, that is, modifying and recompiling the program. Other properties may be changed from within the program while it is running. *Top* is an example of the latter, in that it may be changed while a program is being executed.

CRUCIAL CONCEPT

All controls have **properties**, some of which can be set during the running of a program. Some, however, are fixed when the program is compiled, and cannot then be altered.

Another property, similar to *Top*, is *Left*. This specifies the position of the left-hand side of the form, and it can also be changed during the execution of the program. Two more properties which describe the position of the form are *Height* and *Width*, as mentioned in Section 2. Setting any of these in the properties window will change the corresponding appearance of the form. Similar changes will occur if their values are changed while the program is running. Their names are those of the corresponding variables.

In addition to changing the values of these properties, it is possible, during the execution of the program, to find their current values. This is also straightforward, and using their names as variables will return the current setting for these properties. An example of this is shown below; at the same time, a new event is described.

When a program starts, the form it is based on is made **active**. The method associated with this activation event is *Form_Activate*, and is called at this time. Any code included in this subroutine will be executed at this time. When the code shown below is executed, the initial position of the left-hand side of the form will be displayed.

```
Private Sub Form_Activate()  
    ' Show where the left-hand side of the form is at this time  
    Print "The left-hand side is at "; Left  
End Sub
```

CRUCIAL ACTIVITY

Design your own code fragments and embed them in the subroutine code. Execute the resulting program.

When organising the properties of any control, it is sometimes easier to set up items with similar characteristics. By using the **Categorised** tab (see Figure 2.8), separate properties are grouped for ease of reference. The four properties mentioned above (*Top*, *Left*, *Height* and *Width*) all appear in the **Position** category.

It is also simple, when designing Visual Programs, to make approximate choices for some properties. These may be adjusted according to a user's preferences, and formalised at a later stage. An example of this is when adjusting the size of a form. It may be resized as a window, by a 'click and drag' mouse operation. The *Height* and *Width* properties may be seen to alter correspondingly as this takes place.

Quick test

1. How does Visual Basic assist with program layout prototyping?
2. Name four properties of a typical Visual Basic control.

Section 8

Further control details

In this section, you will learn more about selected controls.

Other controls have already been mentioned in Section 4 – option buttons, labels, text boxes and pictures. Two of these will be described in more detail. The organisation of controls held within a form will also be illustrated by examples. It is assumed that the instructions given below will be followed as they are described; and that suitable notes will also be taken.

The two controls to be described here are **labels** and **text boxes**. These are set up in a form, using the toolbox, and their exact size or positioning will not be important at this stage. When the program is running, the user will be able to change the text shown by the label – it will be organised to match whatever has been typed into the text box. This change will take place when the mouse is clicked on the form.

It is useful to distinguish carefully between these two controls. A label, as you might expect, is for adding a name to some part of the form. It is not intended that a user can

change it during the running of a Visual Program. A text box, however, is intended to display text which may also be altered; although it may be for display purposes only.

Form_Click is the event which will cause this to take place, as in the example in Section 3. It will therefore be necessary to take the value of the text shown in the text box at this time, and assign it to the label being displayed. First, however, the label and text box controls should be positioned on the form.

There are now three controls within the Visual Program. They are easy to select, if any changes are to be made. At the top of the **Properties** window there is a combo box holding the available controls: selecting the control automatically allows its properties to be changed as required. Alternatively, a single click on the control selects its properties in the same way.

The code will be added to the *Form_Click* method; but the effects are not to take place within the form itself, but rather in the label and the text box. The source for the information is the contents of the text box: the **text** attribute is described as 'the text contained in the control', which is what is needed here. It is to be sent to the 'name' of the label: this is the **caption**, and its description is given similarly.

CRUCIAL CONCEPT

The method called when the mouse button is clicked inside a form is *Form_Click*.

The properties of each control show that a caption is a properly common to all of them. Because of this, the use of the identifier *caption* on its own will not be sufficient to distinguish which of the three captions is to be used. It is therefore necessary to be able to identify the components of one control from within a different control. This is done using a full name of any property. This name is constructed by naming the control whose property is to be used, following this with a dot ('.'), and then adding the name of the property. (This will work in any component; so that a full name of this kind may also be used within a component when referring to itself.)

Thus the source to be used is *Text1.Text* (assuming that the default name of the text box is *Text1*). The destination, assuming a similar default name, is *Label1.Caption*. The instruction to be added to the *Form_Click* method is therefore

```
Label1.Caption = Text1.Text
```

There is no other event to be used in this program; this method will carry out all that is needed (see Figure 2.9).

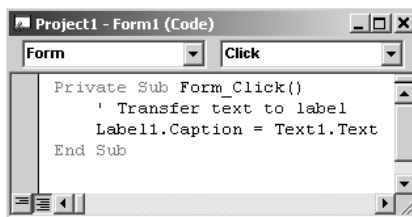


Figure 2.9 Example of method

CRUCIAL ACTIVITY

It is now appropriate to run the program. This is done as suggested in Section 3.

While the program is running, it is obvious that no change takes place in the label until the necessary event (click within the form) takes place. Changing the text in the text box is not enough. It is also easy to see that the *Form_Click* event cannot take place within the

outlines of the label or the text box. An example of the appearance of the program, as it is being executed, is shown in Figure 2.10.

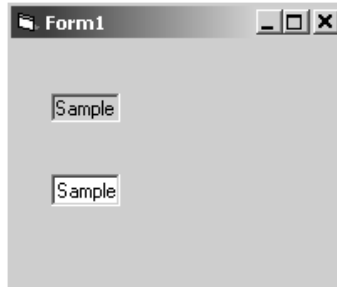


Figure 2.10 event-driven example

The program may be closed down using any of the possibilities given at the end of Section 3.

Quick test

1. How is one control referred to from the code written for another control?
2. Name two control properties which may be changed during the execution of a Visual Program.

Section 9

Appearance of the program code

In this section, you will learn about customising the appearance of a Visual Program.

While you are working with Visual Basic, you can affect the way the code text appears on your screen. There is a default set up, shown in Figure 2.11. You find this from the *Tools* command on the *Options* menu, under *Editor Format*. You can change many of the settings, even in ways that are not useful. (It is **not** a good idea to have green text on a green background, for instance.) In most cases, it is worth leaving everything as it is.

CRUCIAL ACTIVITY

As an example, however, change the *Keyword Text* to an alternative colour. Check the effect this has on the appearance of the code.

This may not appear to be particularly important. It is, however, one of the ways in which programs may be presented more effectively. If it is possible to improve the appearance of anything and make it easier to work with, this is an advantage. It introduces less strain, the users do not tire so rapidly, and consequently are less liable to make mistakes. It is one further aspect of the importance of good interface design and implementation. This, of course, applies also to the products of Visual Programming, as well as the environments in which they are developed.

Quick test

Why might it be good practice to change the colours or sizes of Visual Program code?

Section 10

End of chapter assessment

Questions

1. What is a *Form_Click*, and what can it be used for?
2. How many twips are there in 2 centimetres?
3. Name three fundamental principles of good Software Engineering practice.
4. Which is better for viewing: blue letters on a white background, or yellow letters on a black background? (This should be tested by use of the Editor Format option.)
5. How many categories of control are there?
6. How is the Toolbox displayed?
7. How are controls selected and positioned on a form?
8. What are the keywords which start and end the code for a method?

Answers

1. *Form_Click* is a method within the code of a form which is entered when a mouse click event takes place while the mouse pointer is inside the form's borders. It can be used to hold the code to perform any action appropriate to the program's needs.
2. There are $2 \times 567 = 1134$ twips in 2 centimetres.
3. These include code reproducibility, stability, robustness, reuse, understandability – there are several other possibilities.
4. This is actually a matter of individual perception – the author prefers blue on white, but does not insist that is correct for you. However: did you notice the side-effects?
5. The Help system refers to three categories in Visual Basic 6.0: **Intrinsic controls**, such as the command button and frame controls; **ActiveX controls**, which exist as separate files with a .ocx file name extension; and **Insertable Objects**, such as a Microsoft Excel Worksheet. (This is not meant to introduce new ideas, but to illustrate the use of the Help facility.)
6. By selecting *Toolbox* from the *View* menu, or using the *Toolbox* button on the toolbar.
7. The control is selected from the toolbox and positioned on the form using 'drag and drop'. (This also allows the control's size to be specified.)
8. The method starts with 'Private Sub' and ends with 'End Sub'.

Section 11

Further reading and research

<http://web.nps.navy.mil/~gazolla/is3001/files/VBintro.ppt> holds an interesting introduction to many aspects of event-driven programs centred around Visual Basic.

Kipling, R. (1908). *Just So Stories*, MacMillan & Co. Ltd.