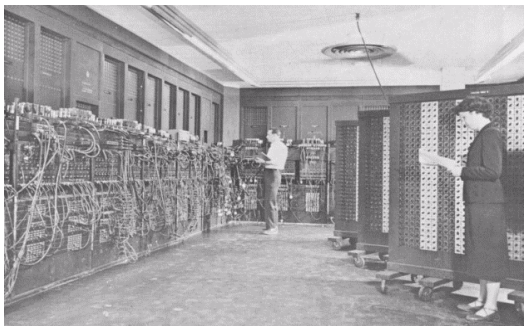
Evolution of Programming

Hard Wiring

The first research computers of the late 1940s were programmed by hard wiring. Cables were plugged and unplugged into huge patch boards to physically alter the electrical circuitry. To program the machine you had to be a highly trained expert in both mathematics and engineering. Later the "stored program" concept was introduced. The machines were programmed in binary with punched holes in paper tape. This was easier, but with all the ones and zeroes to deal with only experts could handle programming.



Machine Code

Further down the track, came machine languages with symbolic type instructions converted by a computer program (an assembler) to binary for the processor. These were called low level languages.

High Level Languages

With third generation computers came more English-like languages like Fortran [***For***mula ***Tran***slation]. These were developed so scientists and people with only a small amount of special training could program computers. They are still hard to use however. Similar languages like Pascal and C++ have been developed to make programming somewhat easier. After a program has been written, but before it can be executed, the programmer's code has to be processed by a computer program called a compiler to translate the programming language commands into binary instructions for controlling the processor. This creates the executable file.

BASIC [Beginners All-purpose Symbolic Instruction Code] was designed to be particularly easy to use. When a program written in BASIC runs, a computer program called an interpreter changes BASIC commands into instructions for the processor, a less complicated process than compiling. It was soon in use by hobbyists and people with very little training. LOGO functions in a similar manner.

These are "procedural languages" in that they follow the methods of the processor and force the user to comply with instructions in a one-after-the-other linear fashion.

The "structured programming" movement introduced special "object oriented" versions of these languages, making coding easier and helping with debugging. It was really just a change in emphasis, code was arranged more logically with the objects it referred to.

Event Driven Programming

More recently, the GUI (Graphical User Interface) OS (Operating System) has provided an ideal platform for event based programming. In these OSs the user (not the programmer) chooses what happens, in what order, and when, by interacting with objects on the screen. The OS detects these "events" and does the appropriate things.

VB is an environment like this where you design the user interface, that is the objects the user sees on screen, and write code to determines what happens when an event like clicking the mouse takes place.

Visual Basic

History

Visual Basic had its origin in the BASIC language. This was designed in the early '60s as a language to use in teaching computer programming to students.

When the Windows operating system was developed, a version of Visual Basic was created to program for this new system. In versions up to 4.0 VB was an interpreted language. Thus early versions of VB had a speed disadvantage compared with programs written in a fully compiled language such as C++. However, programs were created interactively within a windowed development environment, and so they had the advantage of ease of development. Since version 5, VB programs can be compiled into native code before execution, so it no longer suffers as much from problems with execution speed.

VB allows the (more advanced) programmer to interact with the OS via the Windows API (Application Programming Interface). You can exploit the OS functionality to accomplish almost any conceivable task on a PC.

Overview

Visual Basic is organised around *objects* such as forms and controls. Each object (e.g. command button, text box, list box) has its own group of properties (e.g. size, colour, visible, enabled) whose value can be read or set at design time or by your program as it executes.

Programs written with Visual Basic are primarily *event-driven*. That is, once an object initializes, it waits for some event (e.g. a mouse click on a command button control) to occur, which will cause the appropriate segment of code to execute (if the programmer has provided for this).

Developing a Visual Basic program consists (largely) of designing the program's user interface by creating suitable VB objects and writing the code (or "procedures") attached to object events, in order to modify the properties of other objects.

Flavours of Visual Basic

The Visual Basic programming language comes in a number of guises:

- the Visual Basic programming language package (currently version 6) provides a stand-alone program development environment,
- Recent versions of MS Word, Excel and PowerPoint provide the facility to add functionality by using the Visual Basic for Applications (VBA) variant of VB,
- MS Access includes Access Basic, which allows the programmer to create sophisticated databases, and
- VBScript offers the programmer the facility to add functionality to MS Outlook, and include enhanced features in Web-based documents.

Although developing programs in each of these contexts may require the programmers to familiarise themselves with certain application-specific object types and methods, the fundamental VB programming approach is common.

The Visual Basic Object Model

Objects are fundamental building blocks of the Microsoft Office 97 applications; nearly everything you do in Visual Basic involves manipulating objects. Every unit of content and functionality in Office - each workbook, worksheet, document, range of text, slide and so on - is an object that you can control programmatically using Visual Basic. When you understand how to work with objects, you're ready to automate tasks in Office.

Objects, Properties and Methods

Objects are arranged in a hierarchy or object model. The top level object in an application is usually the **Application** object, which is the application itself. The **Application** object contains other objects. Typically, objects are contained in other objects and have objects that are contained in them. For example, the Excel **Application** object contains **Workbook** objects, each of which contain **Worksheet** objects, each of which contain **Range** objects.

Before you can do anything to an object, you must return a reference to the object. To do this, you must build an expression that gains access to one object in the object model and then uses properties and methods to move up or down the object hierarchy until you get to the object you want to work with. To get at the content and functionality of an object, you use properties and methods of that object.

The following table uses properties of the **Range** object of Excel.

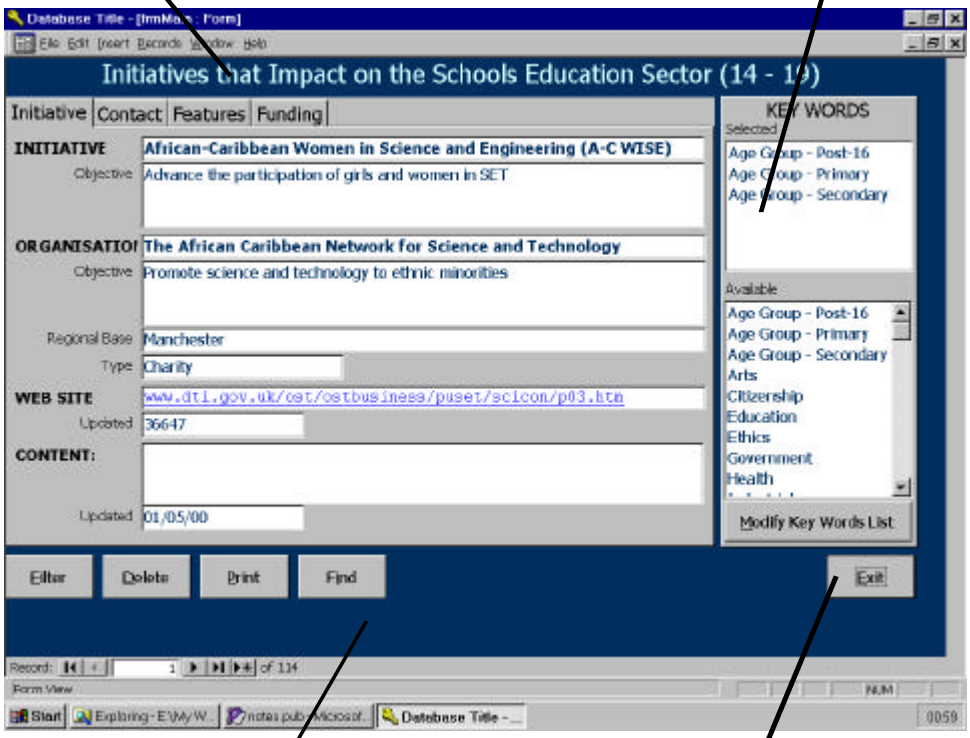
To do this	Use the following code
Set the value of cell A1 on Sheet 1	`Worksheets("Sheet1").Range("A1").Value = 3`
Set the formula for cell B1 on the active	`Range("B1").Formula = "=5-10*RAND()"`
Set the value of each cell in the range C1:	`Range("C1:E3").Value = 6`
Clear the contents of the range A1:E3 on	`Range("A1","E3").ClearContents`
Set the font style for the range named "myRange" (a workbook-level name) to	`Range("myRange").Font.Bold = True`
Set the value of each cell in the range	`Range("Sheet1!yourRange").Value = 3`
Set an object variable to refer to a range	`Set objRange = Range("myRange")`

The **Range** object can represent a single cell, a range of cells, an entire row or column, a selection containing multiple areas, or a 3-D range.

We use Access Basic to refer to properties of controls on forms using a similar approach and syntax (see figure on the next page).

Me![lblTitle].Caption = "Initiatives that .... (14-19)"

Me![lboxkeyWordAll].ColumnCount = 2



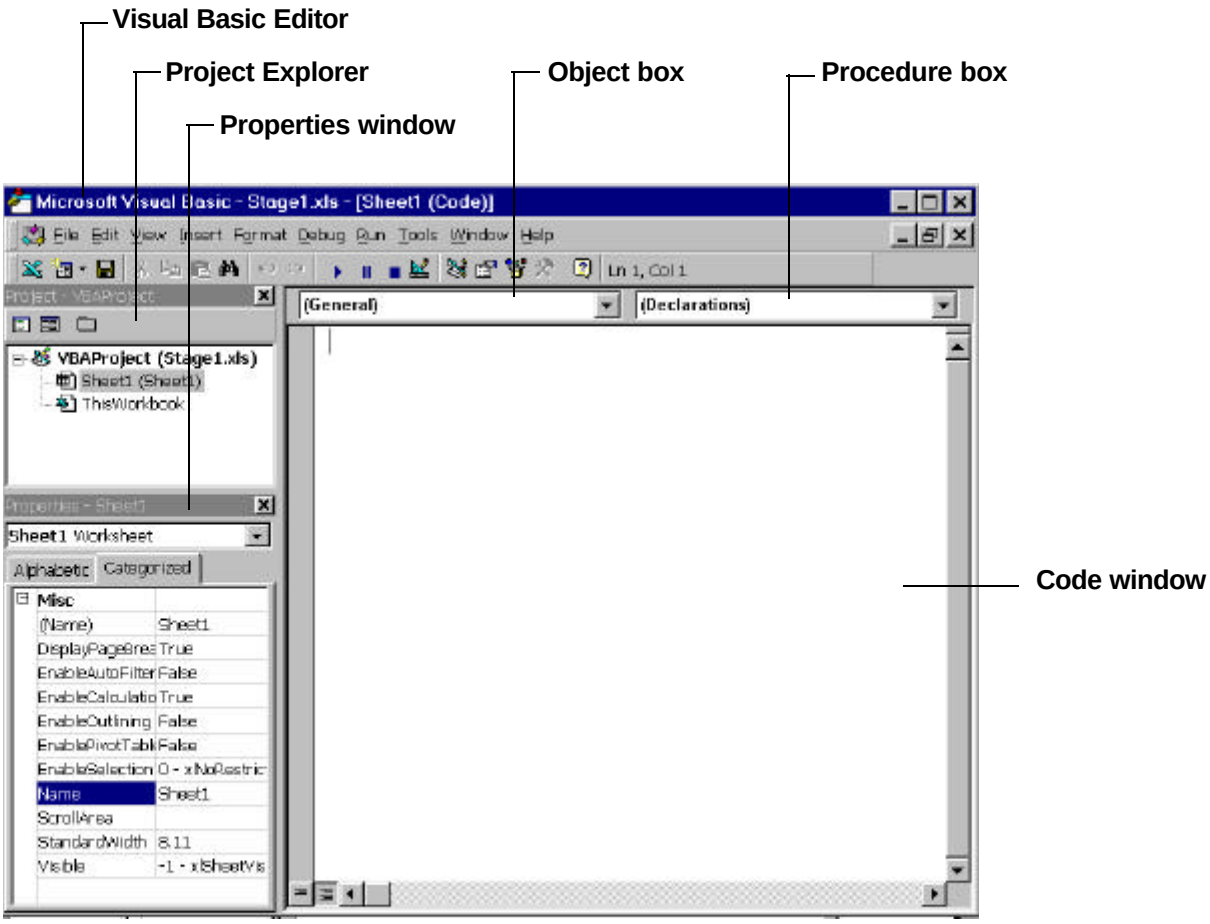
Forms![frmMain].Detail0.Visible = True

Forms![frmMain]![cmdexit].SetFocus

Note that the three expressions listed below are equivalent:

```
Forms!frmMain!cmdExit.Enabled
Forms![frmMain]![cmdExit].Enabled
Forms("frmMain")("cmdExit").Enabled
```

The Visual Basic Editor



For information about a particular window in the Visual Basic Editor, click in the window and then press F1 to open the appropriate Help topic. To see the Help topic for any other element of the Visual Basic Editor, search Help for the name of the element.

The Properties Window

A property is a characteristic of an object, such as the object's colour or caption. You may set a property to specify a characteristic or behaviour of an object. For example, you can set the **ShowSpellingErrors** property of a Word document to **True** to show spelling errors in the document.

You can use the Properties window to set or read the properties of an object at design time. You may also set or read the properties of an object using Visual basic code at run time.

If you don't think you'll be using the Properties window now, you may close it to simplify your work space. You can open it again by clicking the Properties Window in the View menu.

The Project Explorer

All the code associated with a workbook, document, template or presentation is stored in a project that is automatically stored and saved with the workbook., document, template or presentation. In the Project Explorer of the Visual Basic Editor, you can view, modify and navigate the projects for every open or referenced workbook, document, template or presentation.

The Code Window

To view the code in a project, go to the Project Explorer, click the element that contains the code, then click the View Code button at the top of the Project Explorer.

You can navigate the code by using items listed in the Object and Procedure boxes at the top of the window.

In the Object box, click (General), and then click a procedure name in the Procedure box to see a procedure that isn't associated with a specific event.

In the Object box, click an object, and then click an event in the procedure box to see the code that runs when a specific event occurs.