

CHAPTER 1

InFocus

WPL_VBW1001

UNDERSTANDING WORD VBA

Visual Basic, or **VB** as it is affectionately known, is a programming language that was developed many years ago by **Microsoft** to help people program their computers. **Microsoft Office** contains a number of applications that each have a derivative version of **VB** which has become known as **Visual Basic for Applications**, or **VBA** for short.

VBA is an *object-oriented* programming language. It is a language that performs operations by manipulating **objects** which in turn have **methods**, **properties**, and **events**. Each application in the **Microsoft Office** suite, including **Microsoft Word**, has its own specific set of objects. So what better place to start than to examine **Microsoft Word's** *object model* and hierarchy.

In this session you will:

- ✓ gain an understanding of programming in **Microsoft Word**
- ✓ gain an understanding of **VBA** terminology
- ✓ gain an understanding of objects in **VBA**
- ✓ gain an understanding of the **Microsoft Word** object hierarchy
- ✓ learn how to view the **Word** object model
- ✓ learn how to use the **Immediate Window**
- ✓ learn how to work with object collections
- ✓ learn how to set property values
- ✓ learn how to use the **Object Browser**
- ✓ learn how to program with the **Object Browser**
- ✓ gain an understanding of getting good and reliable help for **VBA**.

PROGRAMMING IN MICROSOFT WORD

Microsoft Word has had some form of programming language available to it ever since it was first released. Its current programming language, **VBA**, is a very powerful and versatile

tool. It is not overly difficult to learn and use, but it sometimes can defy logic and comprehension until you really fully grasp its background and what it is primarily designed to do.

Macros versus VBA – What's the Difference?

In a word – none! When Word documents were in their infancy it was possible to program them to perform repetitive procedures. These programs were simply lists of commands from the standard menu that you wanted to run in a particular sequence. This early form of a program was known as a **macro**.

The term **macro** is still used today – indeed you'll see it on the **Developer** tab of the **Ribbon** in **Microsoft Word 2010**. However, these macros are now based on a full-blown programming language, **VBA**, rather than a list of simple commands. So the two terms, **macro** and **VBA**, are interchangeable.

The Real Reasons for Using VBA

The sole justification behind putting a programming language behind an application like Microsoft Word is to extend its capabilities beyond those that you can find on the **Ribbon**. There are several reasons people decide to program in VBA.

Repetition

The first and primary use of VBA is to automate operations especially those performed in a repetitive manner. Let's say every day you need to print every page in a 20 page document (to a **PDF** file of course), or colour every subheading in a document blue, or import your Excel sales data into a table, and chart it on a weekly basis. These are repetitive tasks that can be dead boring or time consuming to do. And, they're perfect candidates for VBA.

Limitation and Interaction

Imagine developing the world's best monthly report, full of facts, tables, graphics and illustrations that demonstrate the sales productivity of several offices around the world. Then imagine handing that work of art to the office casual to adjust the monthly figures only to find they have accidentally deleted the data relating to an entire office! Eek! With carefully crafted VBA coding you can gently steer even the most novice of users through your masterpiece, giving them access to specific areas and precluding them from others, providing them with specialised dialogue boxes (known in VBA as **forms**) and prompting them for key information.

And if all else fails you can even write a simple hangman program in VBA to keep your users amused!

Cross-Application Communication

The Microsoft marketing gurus have told us for years how easy this really is. But in the real world, away from the spin, this is best handled by some VBA programming. Whether it's pulling **Excel** data and charts into a **PowerPoint** presentation, or dropping sales figures into a **Word** document, VBA through its object-orientation allows you to write programs that communicate across applications.

Time versus Effort

Some people actually love VBA because it's fun! Pitching your wit against the computer, getting a program to run without crashing – these are challenges equalled only by trying to land a person on Mars. But from a business perspective they can also be an expensive trap. If you only need to perform an operation (even a complex one) once, then it would be a waste of time to write a program to do it – no matter how clever the program makes you look!

Also, we'd like a tenner for every program written in VBA that could already be done by something that exists currently on the **Ribbon**. We'd be millionaires! Before you sit down to write a VBA program check to make sure that the task can't already be done in other ways – the moral of the story here is have a good grasp of Microsoft Word before you attempt VBA!

VBA TERMINOLOGY

Visual Basic for Applications is a derivative of the programming language **Visual Basic**. Each Office application has its own particular flavour of **VBA** depending on the **objects** and operation of

the application. For example, Microsoft Word works with *documents*, while Microsoft Excel uses *worksheets*. The following notes explain some of the key concepts in VBA programming.

Object Oriented and Procedure Driven

Visual Basic for Applications and its big brother **Visual Basic** are both **object-oriented** programming languages because they work with **objects**. Most of these objects appear on the screen, hence the term '**visual**'.

They are also **procedure-driven** languages using commands and structures from the **BASIC** programming language to bind object statements into workable applications.

Objects, Properties, Methods and Events

In VBA an **object** is anything in an application that you can see and **manipulate** in some way.

For example, you can manipulate a document by adding pages, changing columns, deleting text, and so on. A **document** is therefore an example of an **object**. Just to add a little complexity, **tables** are objects, as are **paragraphs**. These are **child** objects of the **parent** object – the document. This way of organising objects into a hierarchy is known as an **object model**.

Objects can be manipulated in one of three ways. You can:

- change the way an object looks or behaves by changing its **properties**
- make an object perform a task by using a **method** that is associated with the object
- run a procedure whenever a particular **event** happens to an object.

Objects therefore have **properties**, **methods** and **events**.

A real-world example...

Let's look at a simple real-world analogy to get a better idea about **objects**, **properties**, **methods** and **events**. Let's consider a car. It is an **object** because you can see it and manipulate it. Its:

- **properties** are its physical characteristics such as its make, model, colour and so on
- **methods** define what you can do with the car such as reversing, accelerating, turning, stopping and so on
- **events** are the actions that happen to the car that generate an automatic response from the car. For example, if you remove the keys from the ignition while the car's headlights are on (event), most cars will sound a warning alarm (response).

The Active Object

In VBA, **active** describes the object item that you're currently working on.

For example, the document that you're currently using in Word is the **active** document. The object that is currently active is said to have the **focus**.

This is an important concept to understand because most of your VBA programming will be doing something to a particular object. If you don't identify that object correctly you may find that Word shifts focus behind the scenes to a different object and your program seems to go hay-wire.

You'll soon see how this concept of the **active** object works.

UNDERSTANDING OBJECTS

In VBA, an **object** is anything in an application that you can manipulate in some way. For example, your code may apply a theme to a document, change the style of a paragraph, apply

bold to a range of text, maximise a window, set a specific application option, and so on. Each of these items – document, paragraph, range, window and application – is an **object**.

Object Collections

As you know you can have several Word documents open at any time. Each of these documents is a separate **object** that belongs to a **collection**. A **collection** is simply a set of similar objects. For example, Word's **Documents** collection is the set of all open **Document** objects. Because collections are objects themselves, they have their own properties and methods that you can use to manipulate one or more objects in the collection.

The members of a collection are called **elements**. You can refer to an individual element by the object's name or its **index** value (the **index** is the number that Word lists beside the filename in **Switch Windows** in the **View** tab). For example, you can programmatically close a workbook named **Intro.docx** using the following two commands (where **Intro.docx** is the first workbook opened in the current Word session):

```
Documents("Intro.docx").Close or
Documents(1).Close
```

Notice how the **object** (**Documents**) and the **method** (**Close**) are delimited by a full stop (.).

If you don't specify an **element** – such as ("**Intro.docx**") or (**1**) – VBA assumes you are working with the entire collection.

Object Properties

Every object has a defining set of characteristics. These characteristics are called the object's **properties** and they control the appearance and position of the object. For example, the **Window** object has a **WindowState** property that you can use to display a window as maximised, minimised or normal.

When you refer to a property you use the syntax **Object.Property**. For example:

```
Window.WindowState
```

Properties appear in a listing with a property icon .

Object Methods

An object's **method** describes what you can do with the object. For example, you can **save** (**method**) the **active workbook** (**object**).

When you refer to a method you use the syntax **Object.Method**. For example:


```
ActiveDocument.Save
```

Some methods have **arguments** and you use the syntax **Object.Method argument1, argument2, ...**

```
Documents.Save NoPrompt :=True, OriginalFormat :=wdOriginalDocumentFormat
```

The above statement will save each open document in the **Documents** collection in their original format without first prompting the user (unless a document has not been previously saved). Including the two arguments with this statement is optional; if you don't include them Word will prompt the user to save any documents that have been changed since they were last saved. Note, however, that you cannot specify only one of the expected arguments – it's either both or none.

Notice also, in the example above, how the **:=** operator assigns **values** to the named arguments **NoPrompt** and **OriginalFormat**. The names of the valid arguments, plus the **constants** (such as **wdOriginalDocumentFormat**) that can be used with them, are listed in the Help system for each particular method.

The names of the valid arguments, plus the **constants** that can be used with them, are listed in the **Help** system for each particular method. **Methods** appear in a listing with the method icon .

Object Events

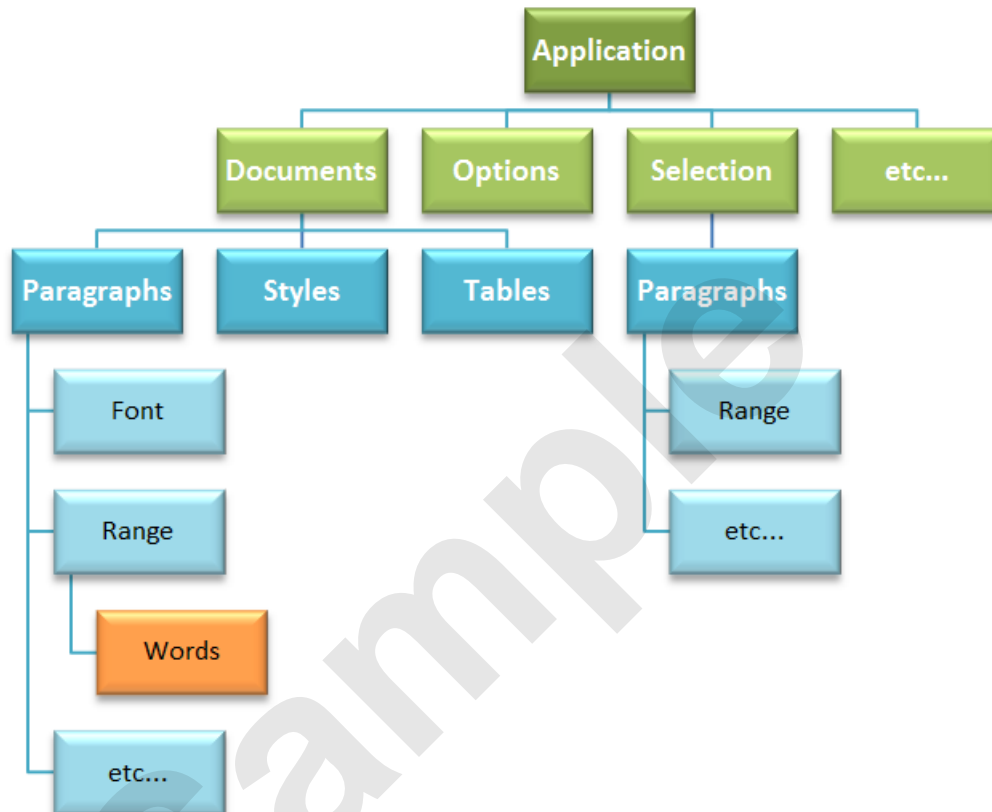
An **event** is something that happens to an object. The opening of a workbook in Excel is an example of an event. Although Excel has an **Open** method that you can use to open a workbook, this **method** only initiates the procedure; the actual process of the file being opened is the **event**.

For example, you may write a procedure (which is called an **event handler**) that will display a message box each time a specific workbook is opened.

THE OBJECT HIERARCHY

Each Office application contains its own set of objects. These objects are arranged in a **hierarchy** with the most general (the *Application* object which refers to the program) at the top. In

Word, this object contains more than 30 objects – we've shown only three below. The lower levels progress through more specific objects, such as **Documents**, **Paragraphs**, **Font** and so on.



For a really clear and complete graphical representation of the *Word Application Object Model Map*, see the web page: <http://msdn.microsoft.com/en-us/library/bb288731.aspx>.

Specifying Objects

To specify an object in the hierarchy you usually start with the uppermost object and add the lower objects, separated by full stops. For example, to insert a new paragraph before a selection of text, you could type:

Application.Selection.Paragraphs(1).Range.InsertParagraphBefore

One of the most confusing aspects of objects and properties is that some properties also act as objects. When you look in Help you will see that the **Application** object has a **Selection** property, but you will also see that **Selection** is an object in its own right – it represents the current selection in a window or pane or the insertion point if nothing is selected.

In other words, lower-level objects in the object hierarchy are really just properties of their parent objects. Therefore, because the **Selection** object implicitly refers to the **Application** object you can reduce the above statement to:

Selection.Paragraphs(1).Range.InsertParagraphBefore

VIEWING THE WORD OBJECT MODEL

The **object model** for Word is a very extensive hierarchy of **objects** and **collections**. In previous versions of Word you could readily find a pictorial representation of the entire model. For some

reason, in **Word 2010**, you can only view pictorial representations of the individual objects of the model. Nevertheless, this is still a great way of finding the methods and properties of an object.


Try This Yourself:

Before starting this exercise ensure that Word has started and the VBA Editor is open...

- 1 From the VBA Editor select **Help > Microsoft Visual Basic for Applications Help** to open the **Word Help** window


- 2 Click on **Show Table of Contents** 

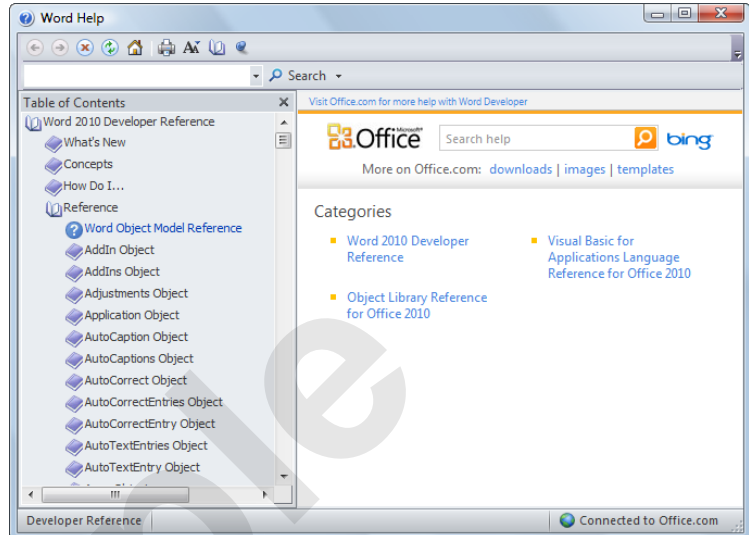
- 3 Click on **Word 2010 Developer Reference** in the **Table of Contents**, then click on **Reference**, to display the list of objects in the model

Objects that appear with a closed book icon , have an additional level of hierarchy...

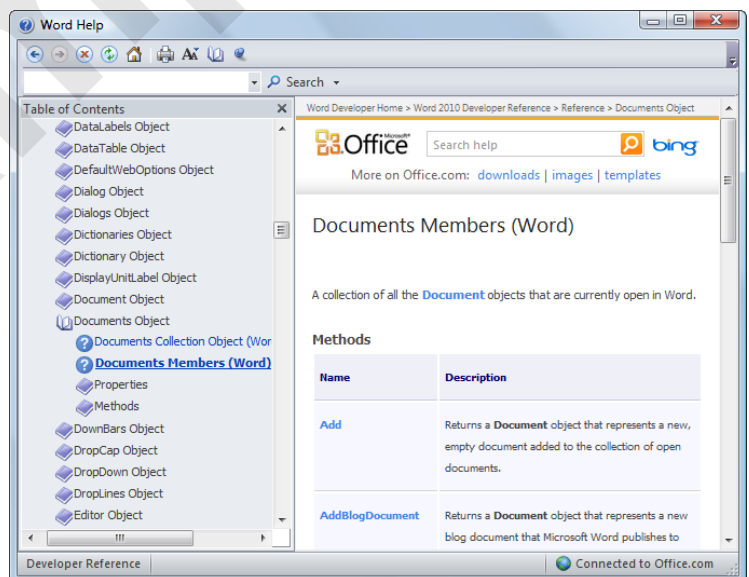
- 4 Scroll to and click on **Documents Object**, then click on **Documents Members (Word)** to display the help

This displays the methods and properties for the Documents object...

- 5 Click on the **Close**  to close the **Help** window




3



4

For Your Reference...

To **access** the **Word object model**:

1. Select **Help > Microsoft Visual Basic for Applications Help**
2. Click on **Show Table of Contents** 
3. Click on **Word 2010 Developer Reference**

Handy to Know...

- It is more important to know **how** to find the object model than it is to memorise the entire structure. Understanding how these objects, methods and properties work together will come with time and experience.
- You can also find the model on Microsoft's **MSDN** website.

USING THE IMMEDIATE WINDOW

The **VBA Editor** has an **Immediate Window** that lets you type instructions and test expressions. It runs statements immediately as if they were run from a procedure. You can also precede

variables and expressions with a **question mark** to perform **what is** operations. For example, **? Documents.Count** asks "How many documents are open?".

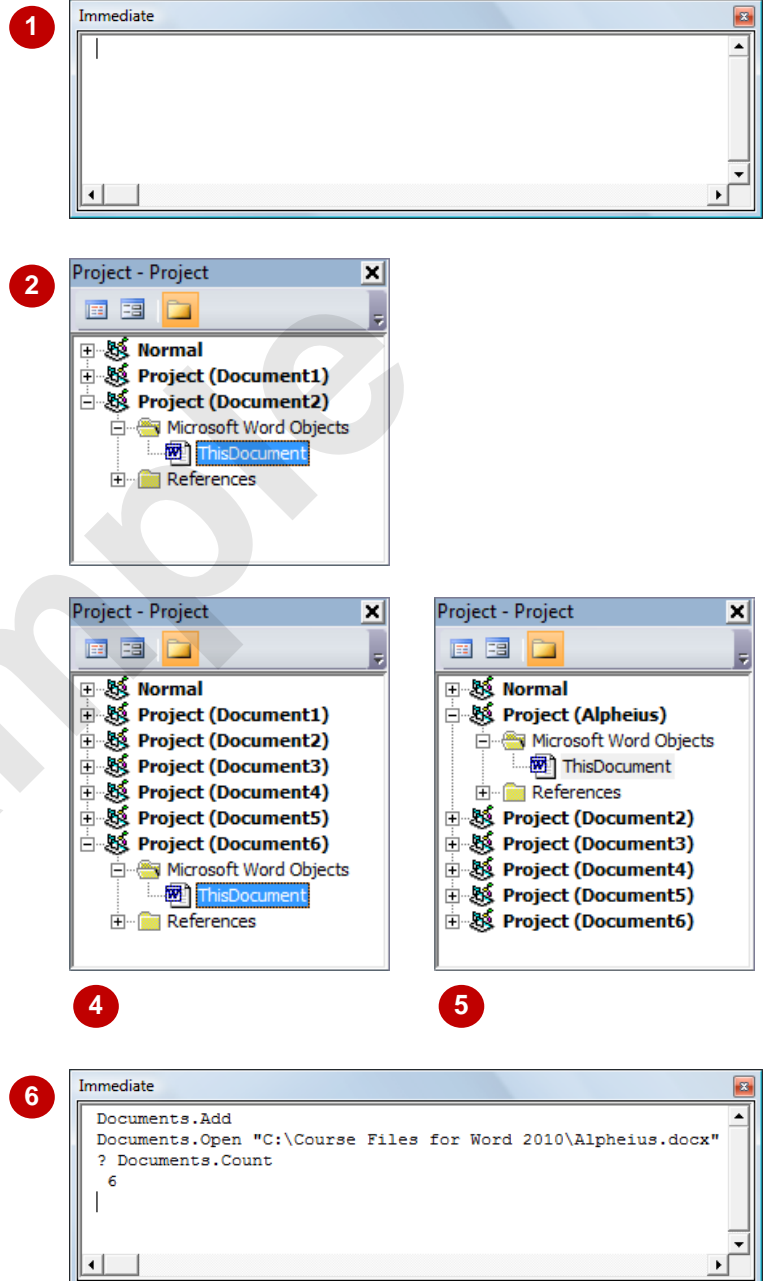
Try This Yourself:

Before starting this exercise open a new blank document and open the VBA Editor...

- 1 Select **View > Immediate Window** to display the **Immediate Window** as a pane in the VBA Editor
- 2 Type **Documents.Add** then press **Enter** to create a new document – as seen in the **Project Explorer** pane
- 3 Click on **Documents.Add** then press **Enter** to run the command again
- 4 Repeat step 3 three more times to open a collection of documents
- 5 Click under the command in the **Immediate Window** and type **Documents.Open "C:\Course Files for Word 2010\Alpheius.docx"** then press **Enter** to open the document called **Alpheius.docx**
- 6 Type **? Documents.Count** and press **Enter**

It will replace Document1 in the list...

This statement counts the number of open documents



For Your Reference...

To **display** the **Immediate Window**:

1. Select **View > Immediate Window**

To **use** the **Immediate Window**:

1. Type a command in the window
2. Press **Enter**

Handy to Know...

- Computer programming requires a much higher degree of accuracy than you may be used to. For instance, if you have typed the file and file path details incorrectly, or if the files are in a different location on your computer, the programming instructions you type will result in an error message.

WORKING WITH OBJECT COLLECTIONS

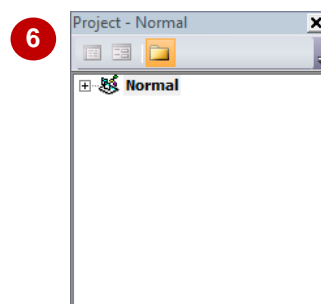
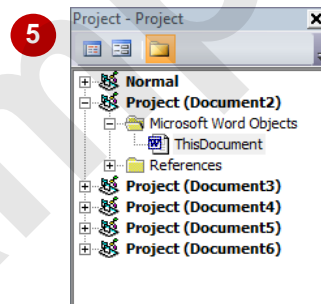
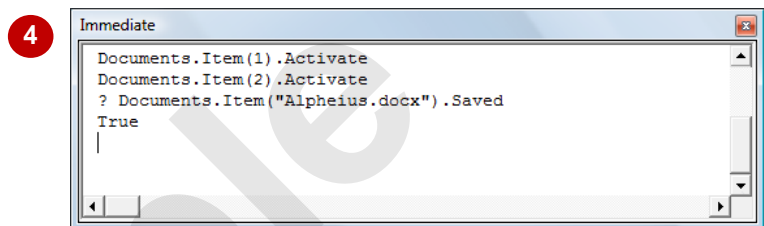
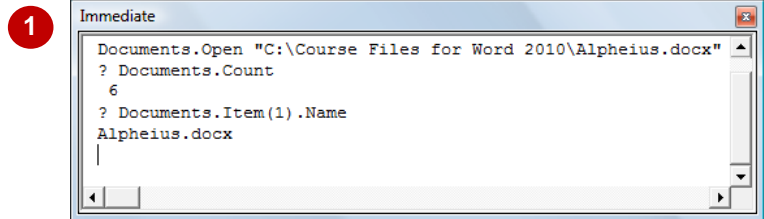
In VBA some objects belong to collections. The **Documents** collection is the set of all the **document** objects open in the **application**. All of the **templates** currently available belong to the

Templates collection. Each **item** in a collection is known as an **element** and must be referenced as part of the collection, either by **name** or by its position in the collection (known as the **index**).

Try This Yourself:

Continue using the previous file with this exercise...

- 1 Ensure the cursor is positioned on a blank line in the **Immediate Window**, then type **? Documents.Item (1).Name** and press **Enter** to return the name of the first document
- 2 Type **Documents.Item(1).Activate** and press **Enter** to make this the active document
- 3 Type **Documents.Item(2).Activate** and press **Enter** to make the second document active
- 4 Type **? Documents.Item ("Alpheius.docx").Saved** and press **Enter** to see whether the file has been saved – *True* if it has, *False* if it has not
- 5 Type **Documents.Item ("Alpheius.docx").Close** and press **Enter** to close the document
The document Alpheius.docx will disappear from the list...
- 6 Type **Documents.Close** and press **Enter** to close all documents – don't save if prompted



For Your Reference...

To **work** with **collections** use the structures:
CollectionName.Item(index or name).Method
CollectionName.Item(index or name).Property = value

Handy to Know...

- You can find out information about the active document by using the object **ActiveDocument**. For example, **? ActiveDocument.Name** will display the name of the active document.

SETTING PROPERTY VALUES

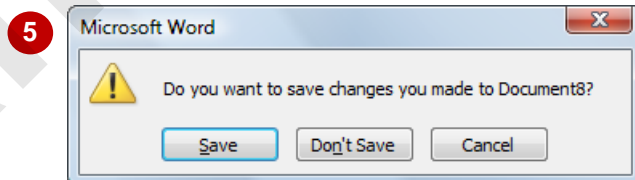
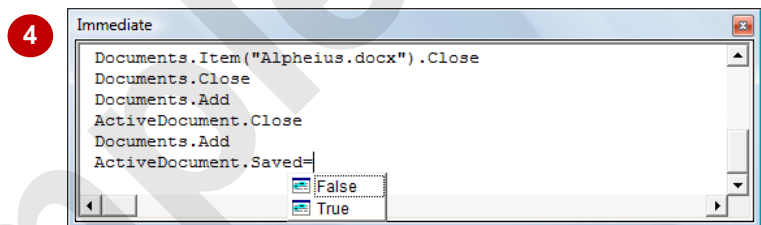
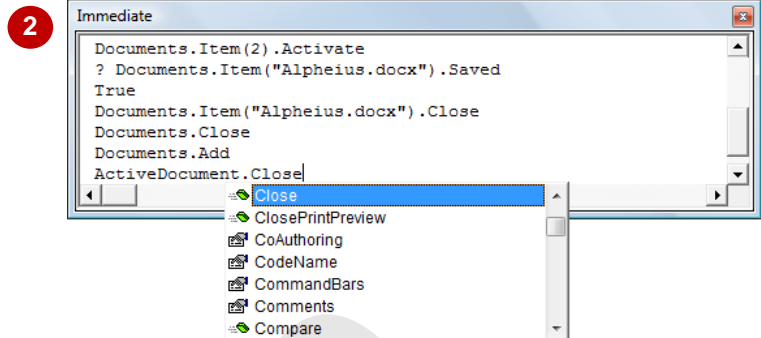
Properties affect the way an object looks or behaves. The properties for objects can be listed by typing the name of the **object** followed by a full stop. VBA includes a built-in **list** system

(known as **Intellisense**) that will list methods, properties and values as you type your command. **Property values** are set by specifying the **object**, the **property**, an **equal sign** and the new **value**.

Try This Yourself:

Before starting this exercise ensure your cursor is positioned in the Immediate Window in the VBA Editor...

- 1 Type **Documents.Add** and press **Enter** to add a document
- 2 Type **ActiveDocument.Close**
Notice how a list of methods and properties appears when you press the full stop or get to the end of a method or property...
- 3 Press **Enter**, then type **Documents.Add** and press **Enter**
- 4 Type **ActiveDocument.Saved=**
Intellisense will list the possible values for the Saved property. Let's use the logical value False to force Word to display a prompt to save the unsaved document when it is closed...
- 5 Type **False** and press **Enter**, then repeat step 2 and press **Enter**
This will close the document but now you'll be prompted to save it...
- 6 Click on **[Don't Save]**



For Your Reference...

To **set** a **property value** use the structure:

Object.Property = Value

Value can be any one of VBA's recognised data types, including: **Numeric values** (e.g. `ActiveDocument.Range.Font.Size = 14`) or **String values** (e.g. `ActiveDocument.Range.Font.Name = "Arial"`)

Handy to Know...

- Properties are fiddly and take a bit of getting used to. Some properties can only be read, while others can be changed. When you change a property you are said to be **setting** its value. When you read a property you are said to be **getting** its value.





USING THE OBJECT BROWSER

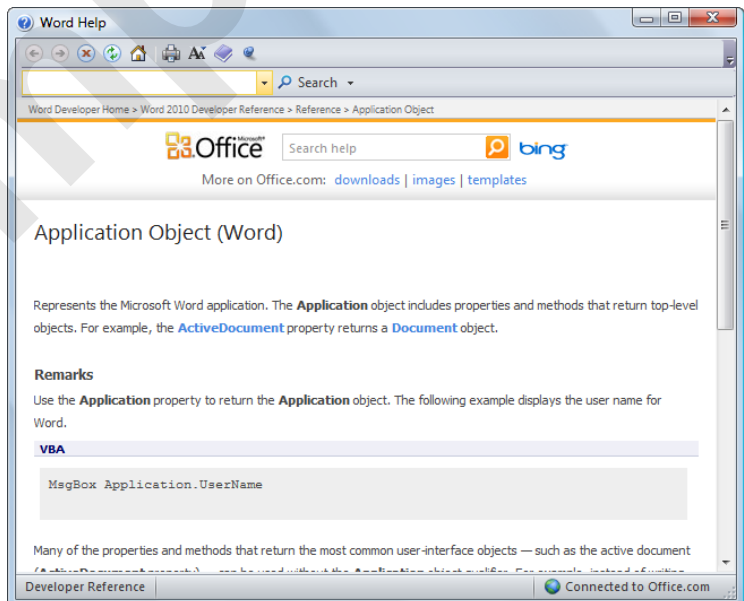
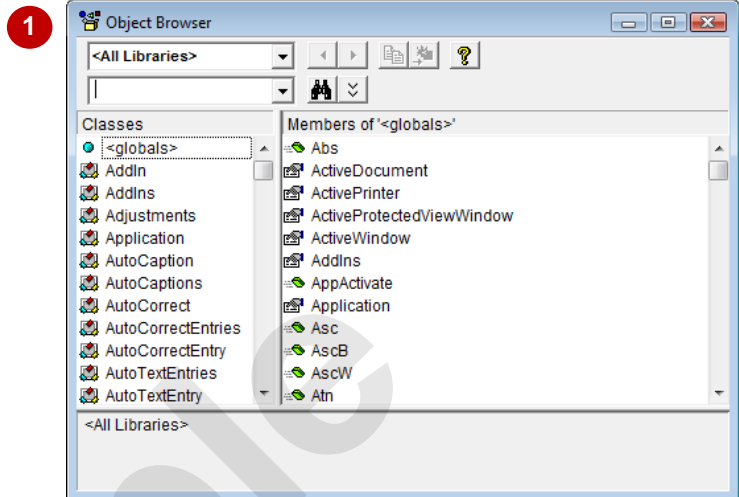
There are literally hundreds of objects, methods and properties available in VBA. Each application in Microsoft Office has its own **object hierarchy** which can be accessed from the **VBA Editor** and

the code modules you develop. The **Object Browser** in the **VBA Editor** allows you to see the object hierarchy and learn more about the relevant **objects**, **methods** and **properties**.

Try This Yourself:

Before starting this exercise ensure you are in the VBA Editor...

- 1 Select **View > Object Browser** to open the **Object Browser** window
- 2 Click on the drop arrow  for **<All Libraries>** and select **Word**
Word objects will be listed under Classes...
- 3 Click on **Application** under **Classes**, then click on **Help**  to display help about the **Application** object
- 4 Click on other **Classes** and **Members** listed in the **Object Browser**, then click on **Help**  to display help about them
- 5 Click on **Close**  to close the **Help** window




For Your Reference...

To **display** the **Object Browser**:





1. Select **View > Object Browser**

To **access Help** for an **object**:

1. Click on the object under **Classes**
2. Click on **Help** 

Handy to Know...

- The **icons** next to the **items** indicate their type:

	Property
	Method
	Event
	Constant



PROGRAMMING WITH THE OBJECT BROWSER

It would be difficult for you to learn what each object does before starting to program. Most people gain a basic understanding of key objects like documents, and from there the knowledge is

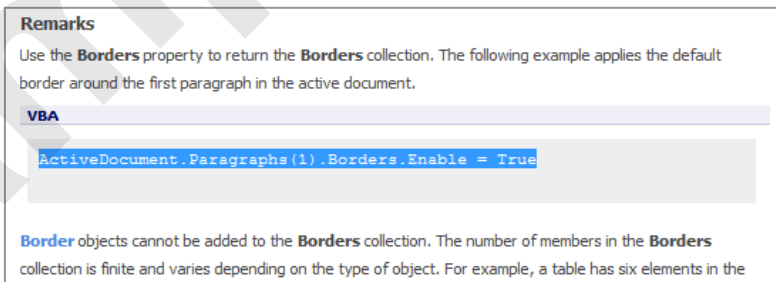
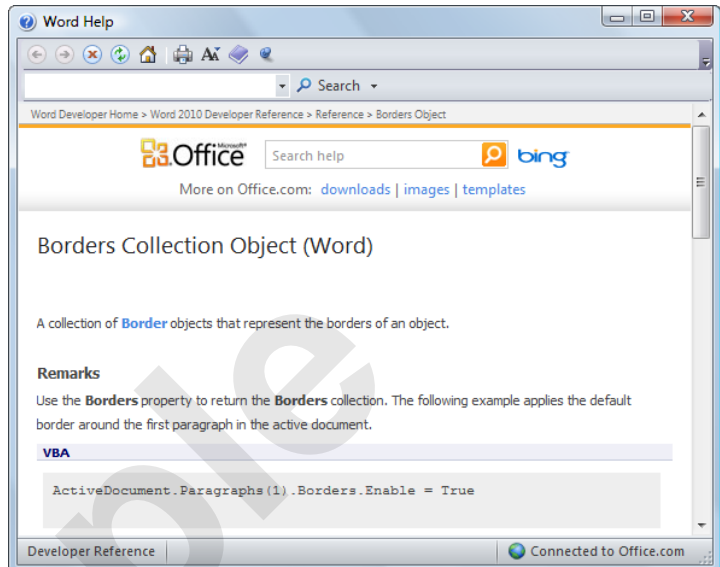
cumulative and on a need-to-know basis. The **Object Browser** lets you browse through the object hierarchies to find out what is the most appropriate object, method or property to use.

Try This Yourself:

Before starting this exercise ensure you have a blank document open and the Object Browser is open in the VBA Editor...

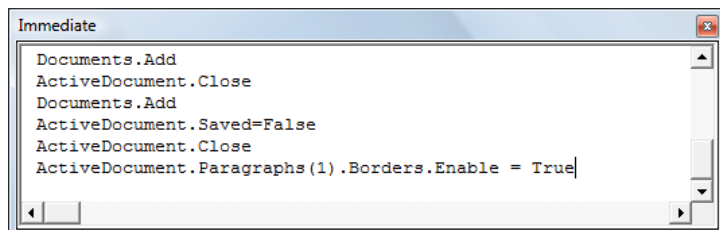
- 1 Scroll down to and click on **Borders** in **Classes**
- 2 Click on **Help**  to display the help about the **Borders** object
- 3 Select the text as shown
You may need to maximise the Help window...
- 4 Press **Ctrl** + **C** to copy the text, then close **Help**
- 5 Click in the **Immediate Window** and press **Ctrl** + **V** to paste the text from **Help**
- 6 Press **Enter** to run the command, then close the **Immediate Window** and the **Object Browser**
- 7 Click on **View Microsoft Word**  to see the document – a border will appear around the first paragraph

2




3

5



For Your Reference...

To **use** the **Object Browser** to **program**:

1. Click on the desired **object** or **member**
2. Click on **Help** 
3. Locate the appropriate code
4. Copy and paste it into your code module and modify as required

Handy to Know...

- You may not find what you want immediately but explore the **classes** and **members** for ideas until you find a similar example, then adapt the code as needed.

THE BEST VBA HELP AVAILABLE

If you think we're talking about the help system built into VBA by Microsoft you'd be hugely mistaken. The Help system is good but it is frustrating to work with. Sometimes it's difficult to

find what you want; sometimes it's too simplistic while other times it's too complex. The best source of help for VBA comes from your peers and the huge community of programmers.

Fellow Programmers Are Your Best Buddies

Chances are whatever you're trying to do in VBA someone somewhere has already done it. So why not ask them for help or information?

Unlike a lot of other professions, most programmers are often only too happy to share what they have done. In the programming world egos are huge and programmers love to show how clever they are at fixing problems – either theirs or yours.

Many programmers are also quite happy for you to use their coding in what you are trying to do. Of course you shouldn't accept this as a given. If you do use other programmer's coding then you should check that they have allowed this to happen and if it's particularly good and helpful then you should also acknowledge the source of the code – it's still most uncool and indeed unethical in programming circles to claim the work of some other as your own! You'll soon learn how to document your VBA code - a nice little thank-you to the source of coding you have used is usually greatly appreciated, even if chances are it will probably never be seen by another living soul!

Making Contact With Fellow Programmers

There is no-one out there who was born an instant programmer – they all had to learn their crafts and gather their skills and knowledge through study, or trial and error, or from others. Many programmers are only too happy to give back to the broader community and have set up forums or blogs to do just that. Others regularly contribute to forums to moderate or answer questions.

Other programmers write books chock full of sample code for you to use and adapt as you need.

What's The Quickest Source of VBA Help?

Google! Or any other search engine of comparable ability. Forget wasting your time with VBA's help system unless you are prepared to receive a single-source answer to your problem or spend ages navigating through tedious and often cryptic layers of the help system.

The quickest solution to any VBA problem is to type your request into a search engine. For example, when writing this course we entered "*vba runtime error 1004*" (this was the error that appeared earlier in one of the exercises) into Google and got almost 450,000 results in less than half a second. Try it yourself – chances are you'll get even more results as more information is made available on the Web.

If you really want to give it a workout try statements like "*creating a new workbook in vba*", or "*getting the current user name, vba*". Chances are you'll end up with thousands of results many containing sample code that you can quite easily adapt and even learn from. The wheel's already been invented and is spinning at a million miles an hour!

Search engine results will often lead you to quality forums for programmers such as that available through Microsoft's own **MSDN** (*Microsoft Developer Network*). Some of these you'll need to pay for but the majority are free for you to use.

Caveat Emptor

Of course search engine results are only as good as the quality of the content. There's a lot of chaff amongst the grains of wheat and you'll need to do your own sorting. For example, when we searched on "*vba runtime error 1004*" one of the entries, as well as being technically incorrect, was simply blatant promotion of a product to fix registry entries. You'll learn with time what the trusted sources are and what are the dogs.