

The Object Model

Introduction to the Object Model

The object model is a way of referring to individual parts of an Office application. At the "top" of the model is the application itself (e.g. Excel, Word, Outlook, PowerPoint). The application will be composed of Menus and Toolbars, Documents (in Word), Workbooks (in Excel), Slides (in PowerPoint), user-defined settings (e.g. header, footer) and so on. Many of these components will be composed of sub-components: Menus will likely have sub-menus, workbooks will be composed of worksheets, documents will have sections, and so on. These sub-parts may in turn have sub-sub-parts. A worksheet will be composed of individual cells, for example. And these sub-sub-parts may have sub-sub-sub-parts. A cell, for example, is defined by its font-type, font-colour, pattern colour, pattern type, formula and so on.

The object model is a way of representing and working with the hierarchy of objects that make up a complete application.

Object Model syntax

The syntax of the Object Model mirrors the syntax used to describe files and folders. Consider the file "C:\Documents & Settings\JSmith\My Documents\VBA\test.doc". We can tell by looking at that folder and filename that there is a C:\ drive and the C:\ drive contains a folder "Documents & Settings". Within that folder is a sub-folder "JSmith". Within that subfolder is another subfolder "My Documents", then "VBA" and within the "VBA" subfolder is a file "test.doc".

The object model uses a very similar representation. One difference is that instead of using a backslash "\" the object model uses a period "." So, for example, in the Object Model, we refer to the worksheet "accounts" in the workbook "Dept.xls" in this way: `Workbooks("Dept.xls").Worksheets("accounts")`. If we wanted to refer to the range of cells "A5:B7" within the "accounts" sheet we'd use the following notation:

```
Workbooks("Dept.xls").Worksheets("accounts").Range("A5:B7")
```

And if we wanted to refer to the background color of the cells in the above range we'd use

```
Workbooks("Dept.xls").Worksheets("accounts").Range("A5:B7").Interior.ColorIndex
```

Visual Basic can help you when writing these expressions. It does that by giving you prompts as to what are valid things to type given what you've already typed.

Object model - Intellisense - walkthrough

We will show the prompts that Visual Basic gives for the preceding example. This will give good insights into the Object Model.

We begin by typing `Workbooks(`

After we have typed that Visual Basic shows the following

```
Workbooks (  
_Default(Index) As Workbook
```

Visual Basic is asking for an Index. The index specifies which workbook - of the collection of Workbooks that might be open - we are interested in. We can specify the workbook by its name "Dept.xls" (or a number - 1 would mean the first workbook). We type "Dept.xls" followed by a closing bracket and then a period. [Remember the period is like the \ separator with files and documents.]

Visual Basic will then show the following

```
Workbooks ("Dept.xls") .
```



A list of items has appeared in a pop-up menu. The list shows all the things that you can do with a Workbook. The pop-up is "context-sensitive" and shows only those things relevant to the particular type of object to the left of the period. Microsoft calls this behaviour "Intellisense".

You may notice there are two types of icons in the list. One type relates to "properties" and the other to "methods". We'll leave an explanation of properties and methods till later.

The items in the list are in alphabetical order. As you type letters after the period the list scrolls to show the first item that matches the letters you have typed.

We want to access one of the Worksheets within the Workbook so let's type a W after the period to make the list scroll to the W's.

```
Workbooks ("Dept.xls") .w
```



WebOptions happens to be the first W that is relevant to a Workbook object. We're not interested in that but in Worksheets - which is a bit further down the list. We type an "o" to scroll the list down to Worksheets.



To save us typing the rest of the word we press the TAB key and the rest of the word is automatically filled out.

```
Workbooks ("Dept.xls").Worksheets|
```


Then we need to type "(".

```
Workbooks ("Dept.xls").Worksheets (
    _Default(Index) As Object
```

Visual Basic is asking us to specify which - of the collection of Worksheets in the Workbook - we are interested in . We want the "accounts" sheet so we type that in and a ")" and period.

```
Workbooks ("Dept.xls").Worksheets ("accounts").
```

Unfortunately, at this point, Intellisense "runs out of sense". It should tell us what things you can do with a Worksheet, but, unfortunately - we get nothing.

 <p>Question</p>	<p>Can you suggest why this happens? Hint: Look carefully at what happened earlier when we were specifying the Workbook. Did VB do anything different with the Workbook than with the Worksheet?</p>
--	--

We can force Intellisense to continue to give us prompts. The reason it doesn't currently give prompts is because it doesn't recognise that each Worksheet within a collection of Worksheets is in fact a Worksheet. We add a line to force Intellisense to recognise this fact.

```
Dim ws As Worksheet
```

```
Set ws = Workbooks("Dept.xls").Worksheets("accounts")  
ws.
```



The line that starts with Dim tells Visual Basic that ws will refer to a Worksheet.

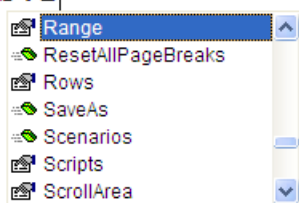
On the next line we "point" ws to the particular worksheet we're interested in.

On the last line we type ws and period and - hey presto! - Intellisense is working again: It is telling us what things we can do with a Worksheet.

We want to work with a particular Range of cells within the worksheet so we type the first letter - r - of Range.

```
Dim ws As Worksheet
```

```
Set ws = Workbooks("Dept.xls").Worksheets("accounts")  
ws.r
```



We press the Tab key and then "(".

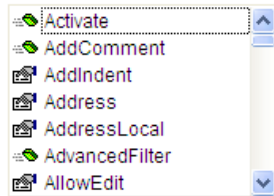
```
Dim ws As Worksheet
```

```
Set ws = Workbooks("Dept.xls").Worksheets("accounts")  
ws.Range(  
    Range(Cell1, [Cell2]) As Range
```

Visual Basic asks us to specify which cells make up the range. Ranges are extremely important and useful in Visual Basic because very often we will want to refer to cells either individually or in some grouping. Fortunately Range provides great flexibility in specifying which cells we are interested in. We'll leave a detailed discussion of that till later. For the time being we will specify the range as "A5:B7". After that we type ")" and period and Intellisense shows us our options with Ranges.

```
Dim ws As Worksheet
```

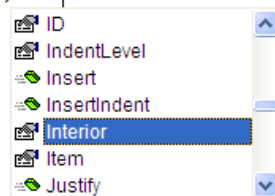
```
Set ws = Workbooks("Dept.xls").Worksheets("accounts")
ws.Range("A5:B7").
```



In our simple example we want to set the background colour of the cells to Red. The background colour is a property of the interior of each cell. So we need to scroll down to Interior in the list.

```
Dim ws As Worksheet
```

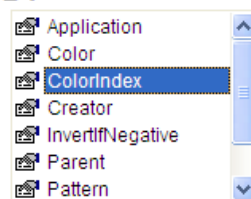
```
Set ws = Workbooks("Dept.xls").Worksheets("accounts")
ws.Range("A5:B7").I
```



We reached the Interior by pressing the Down-Arrow key once the pop-up had scrolled to the first "I". Press the TAB key and then the period.

```
Dim ws As Worksheet
```

```
Set ws = Workbooks("Dept.xls").Worksheets("accounts")
ws.Range("A5:B7").Interior.
```



Choose ColorIndex, press TAB and the equals sign

```
Dim ws As Worksheet
```

```
Set ws = Workbooks("Dept.xls").Worksheets("accounts")
ws.Range("A5:B7").Interior.ColorIndex =
```

It would be good if Intellisense now told us about our color choices - but it doesn't. We just need to know (but more likely - find out by recording a macro) that the ColorIndex that represents red is 3. We type in 3 and finish.

```
Dim ws As Worksheet
```

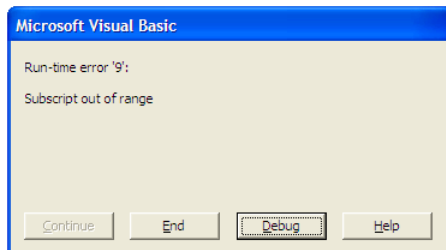
```
Set ws = Workbooks("Dept.xls").Worksheets("accounts")
ws.Range("A5:B7").Interior.ColorIndex = 3
```

When you run this piece of code Visual Basic will set the background colour of A5:B7 to red in the "accounts" worksheet in the "Dept.xls" workbook.

If you don't have the workbook "Dept.xls" open at the time or if the workbook doesn't contain the worksheet "accounts" then Visual Basic will give an error. Visual Basic error messages are often fairly generic. In our example the error message would be :

```
Dim ws As Worksheet
```

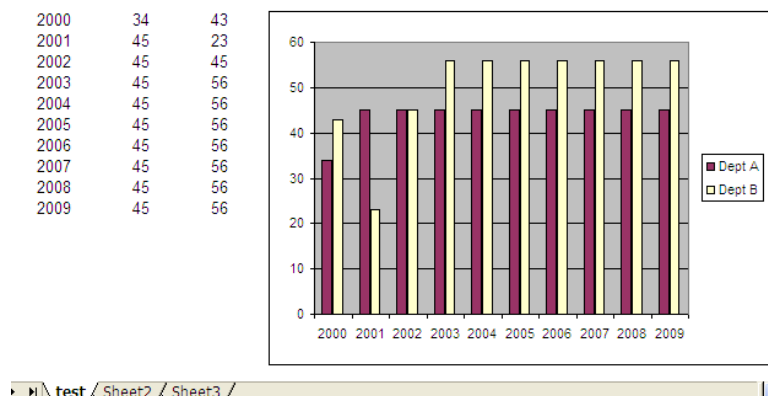
```
Set ws = Workbooks("Dept.xls").Worksheets("accounts")
ws.Range("A5:B7").Interior.ColorIndex = 3
```



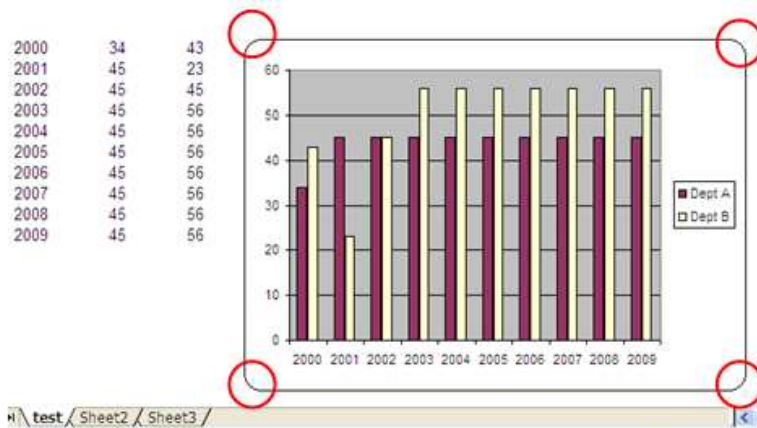
Object model - Intellisense - another walkthrough

In this example we will change the presentation of a chart.

The chart currently looks like this:



We will change the chart so that it has rounded corners so that it looks like this.



As before we begin by forcing Intellisense to show us what we can do with a worksheet.

```
Dim ws As Worksheet
Set ws = Worksheets("test")
```

The chart is on the worksheet named "test" so we set ws to point to the "test" worksheet. Notice that we haven't specified which workbook we want. If we leave the workbook unspecified then Visual Basic uses whichever is the currently active workbook.

Next we use Intellisense to see whether Charts are available within a worksheet.

```
Dim ws As Worksheet
Set ws = Worksheets("test")
```



There aren't any Charts but there are ChartObjects. So let's choose ChartObjects.

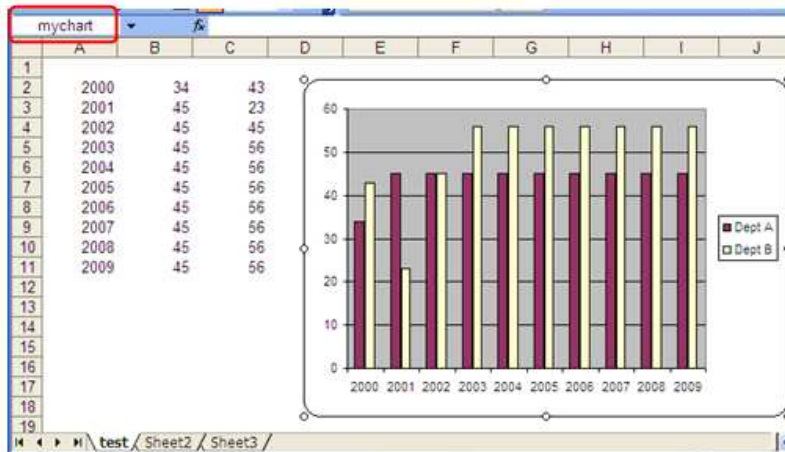
```
Dim ws As Worksheet
Set ws = Worksheets("test")
```

```
ws.ChartObjects(|
  ChartObjects([index]) As Object
```

Visual Basic is asking which ChartObject of the collection of ChartObjects we want. In this case we could simply use the number 1 since we know there is only one

ChartObject on the worksheet. But we'll use an approach that works more generally and refer to the ChartObject by name.

So we need the name of the ChartObject. To find the name go back to Excel, hold the Ctrl key down and left mouse click on the border of the ChartObject. The name of the ChartObject will appear in the name box. [You can change the name here if you need to.] The name is "mychart".



Now that we know the name of the ChartObject we can type that in and then type ")" and period.

```
Dim ws As Worksheet
Set ws = Worksheets("test")

ws.ChartObjects("mychart").|
```

As before with Worksheets Intellisense doesn't tell us what we can do with the ChartObject - We'll use the same trick to force it.

```
Dim ws As Worksheet
Set ws = Worksheets("test")

Dim co As ChartObject
Set co = ws.ChartObjects("mychart")
```



We have declared co as a ChartObject. Then we set co to point to the ChartObject we're interested in. Intellisense now works as we want and shows us what we can do with a ChartObject.

We want to round the corners so we look for "Corner" or "Round" in the popup. We find "RoundedCorners".

```
Dim ws As Worksheet
Set ws = Worksheets("test")

Dim co As ChartObject
Set co = ws.ChartObjects("mychart")
```




Choose RoundedCorners and press the TAB key. Then press the = key.

```
Dim ws As Worksheet
Set ws = Worksheets("test")

Dim co As ChartObject
Set co = ws.ChartObjects("mychart")
```

```
co.RoundedCorners =
```



Intellisense shows we can set the RoundedCorners property to True or False. We set it to True and our macro is complete.

```
Sub SetChartCorner()
    Dim ws As Worksheet
    Set ws = Worksheets("test")

    Dim co As ChartObject
    Set co = ws.ChartObjects("mychart")

    co.RoundedCorners = True
End Sub
```

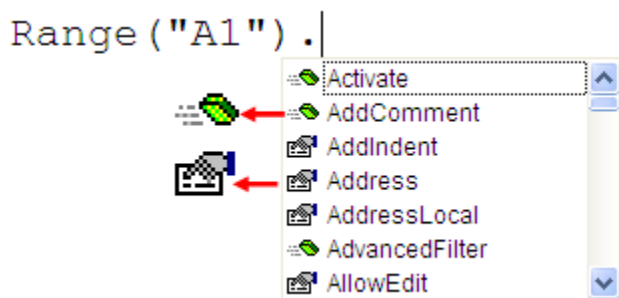
Properties and Methods



Properties and Methods are part of the Object Model. We will look now at properties and methods and explain what they are. Simplifying somewhat, a property is a state that you can **set** or **read** whilst a method is something you can **do**.

To illustrate properties and methods we'll leave the world of computers for a moment and enter the world of people. Consider a person as being an object with properties and methods. Person.Mood would be a property of the person - it is a measure of their state. Person.Shout would be a method since it is an action they take. And, taking the analogy one step too far, perhaps the person is running a code segment that looks like this ..

If Person.Mood = "unhappy" then Person.Shout

Following is a screen snapshot of the Intellisense pop-up that appears after you type Range("A1").



The pop-up shows the things you can do with a Range object. The icon that looks like a flying green brick  represents a **Method**. The icon that looks like a hand over an envelope  represents a **Property**.

For example, you can add a comment to the "A1" cell by calling the AddComment **method** of the Range object. You would do that this way:

```
Range("A1").AddComment "This is a comment"
```

In contrast in Range("A1").Formula, Formula is a **property** that relates to the Formula in "A1".

The Formula property can be used to set (write) or read the formula.

Consider the following, for example

```
Range("A1").Formula = "=SUMPRODUCT(A2:A5,B2:B5)"
```

This would set the formula in "A1" to "=SUMPRODUCT(A2:A5,B2:B5)"

The following line would read the formula in A1 and show it in a Message box.

```
MsgBox Range("A1").Formula
```