

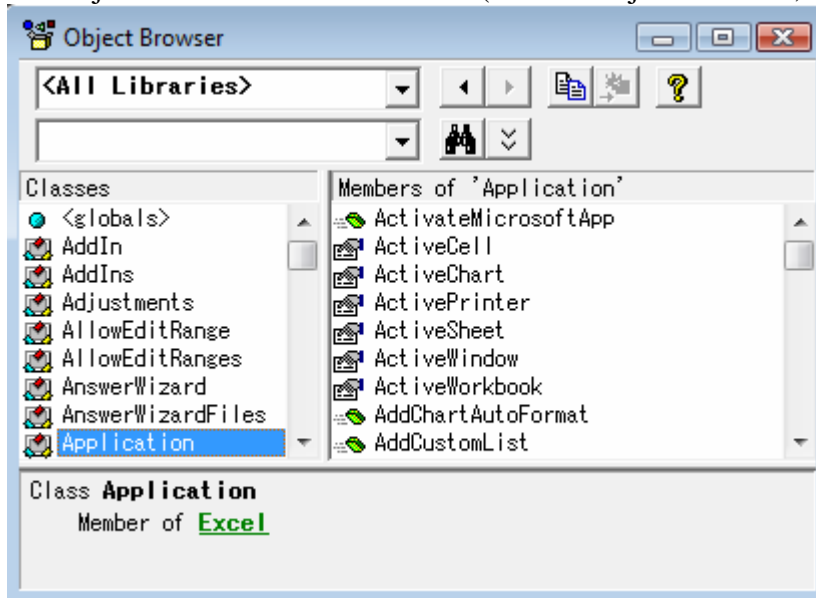
IFA/QFN VBA Tutorial









Notes prepared by Keith Wong

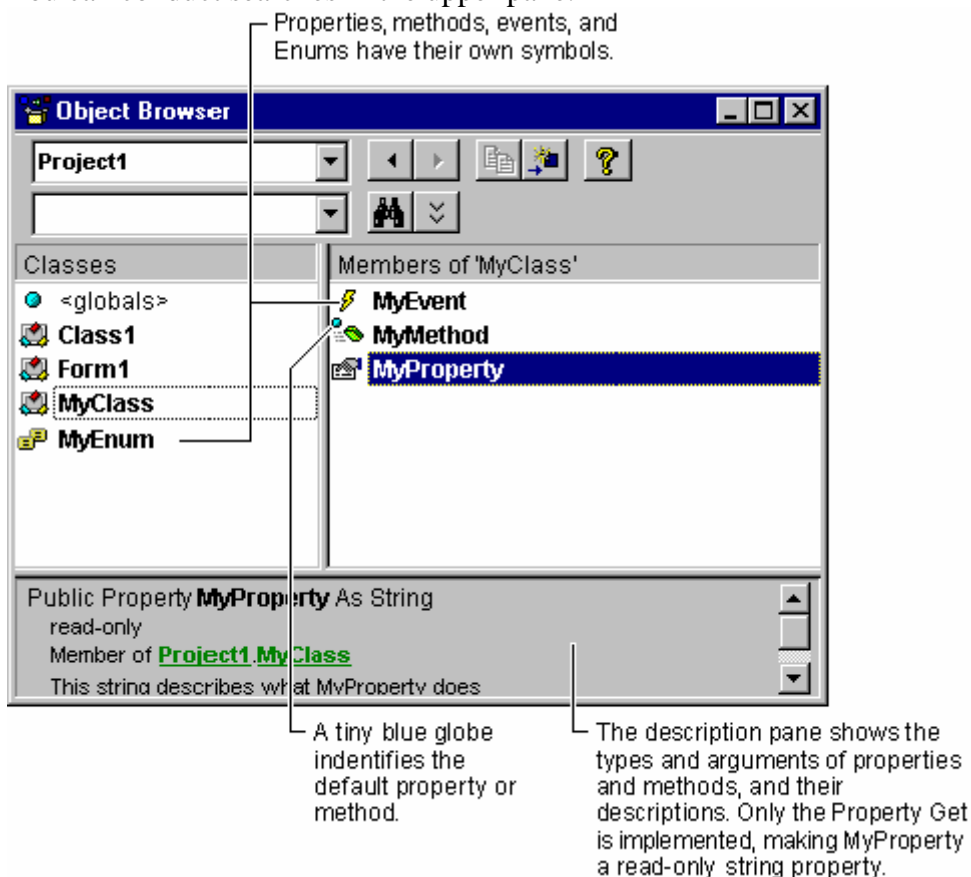
Chapter 5: Excel Object Model

5-1: Object Browser

The Excel Object Model contains thousands of pre-defined classes and constants. You can view them through the Object Browser in the VB editor (View > Object Browser; or press F2).



You can browse through the classes , enums , types , modules  in the left pane, while seeing the properties , methods , events , constants  in the right pane. You can also press F1 on each item selected to see its help conveniently. In the lower pane you can view the descriptions of the selected item. You can conduct searches in the upper pane.



5-2: The Excel Object Model

The classes in the Object Browser form the Excel Object Model. They are defined by Excel so that you can manipulate Excel workbooks, worksheets, etc. using them.

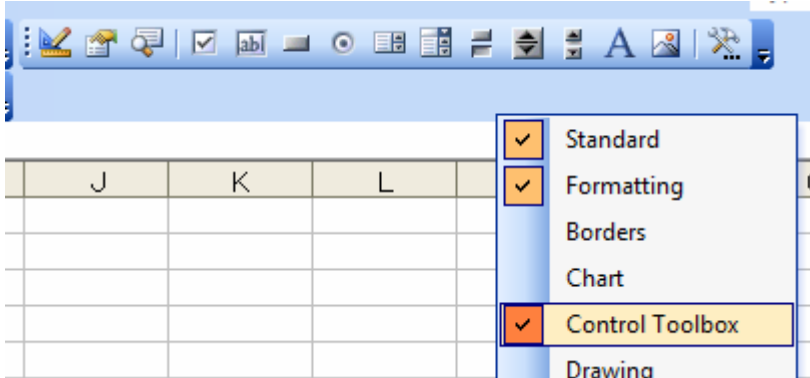
Since there are too many classes and defined objects in the Excel object Model, we cannot discuss them one by one. We can just give a brief introduction on some of the commonly used objects and classes. They are Application, Workbook, Worksheet and Range.

5-3: The Application Object

The Application object, belonging to the Application class, is the root of all the excel objects. It contains a lot of information about the current Excel application, and provides many methods to manipulate Excel in a macroscopic way, such as opening files, executing another application, fetching workbooks, worksheets, current selected cells, and so on.

Let's try! Corner

- 1 Find Control Toolbox and add it to the menu bar.



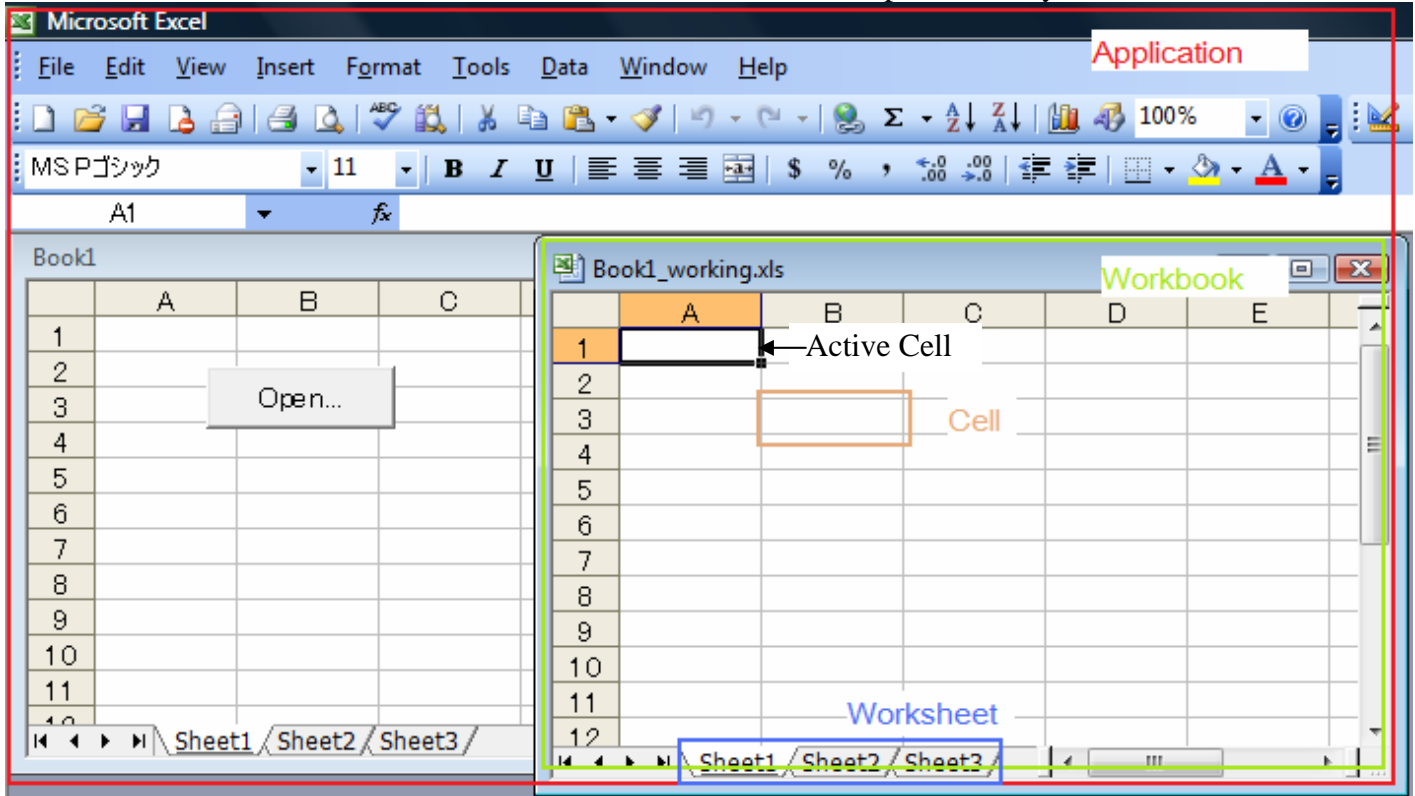
- 2 Insert a command button. Right click the button and choose Properties. Rename the button to btnOpen and change the caption to "Open...".
- 3 Double-click the button. Open Object Browser. Browse to the Application object.
- 4 Browse through the Object Browser and try to find a way to program the button to show the "Open File" dialog box to open an existing workbook.
- 5 Finish your code and go back to the worksheet. Click Exit Design Mode button.



- 6 Test the button.

5-3-1: Access to Workbooks, Sheets and Cells

First we should familiarize ourselves with the structure of an Excel spreadsheet system.



In VBA, using the Application object as the root, we can access workbooks, worksheets and cells easily. A property of the Application object is Workbooks, which returns a collection of opened workbooks in the application. For each workbook, there is a property named Worksheets, which in turn return a collection of worksheets in the workbook. For each worksheet, you can use a property named Cells to return a Range object of all the cells in the worksheet, or the Range property to return a specific range of cells. The value of a cell can be accessed by the value property, which is the default property of the Range class if only one cell is included in the Range object.

When you use the Cells property, you can specify the row index and column index to return a specific cell. An important remark here is that all the indices used in the Excel Object Model are 1-based. For example, Cells(3, 2) selects the cell "B3".

We can access any cells in opened workbooks by means of the above properties. For example, to access the cell B2 in Sheet2 (which is the 2nd sheet) of Book1 (which is the 1st workbook) we can write one of the followings:

```
Application.Workbooks(1).Worksheets(2).Cells(2, 2)  
Application.Workbooks("Book1").Worksheets("Sheet2").Range("B2")
```

or any other combinations of the index form and the name form. You can also omit the Application object, so that the followings are equivalent.

```
Workbooks(1).Worksheets(2).Cells(2, 2)  
Workbooks("Book1").Worksheets("Sheet2").Range("B2")
```

The Application object also provides easy access to the active workbook, worksheet, and cell. They are respectively the `ActiveWorkbook`, `ActiveSheet` and `ActiveCell` properties. You can also omit the Application object when accessing these properties. First make sure that you activate the correct cell.

```
Workbooks("Book1").Activate
Worksheets("Sheet2").Activate
Cells(2, 2).Activate
```

Then all the followings are equivalent.

```
ActiveCell
ActiveSheet.Cells(2, 2)
ActiveWorkbook.Worksheets("Sheet2").Range("B2")
```

You can also omit the `ActiveSheet` or `ActiveWorkbook`:

```
Cells(2, 2)
Range("B2")
```

In fact there are three more properties in the Active series. They are `ActiveChart`, `ActivePrinter` and `ActiveWindow`. You may find out more from the help file.

Let's try! Corner

Construct a 20×20 upper triangular matrix with all the upper triangular entries as 1, including the diagonal entries. Specifically, construct the following:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
3	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
4	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
5	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
6	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
7	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
8	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1
9	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
10	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1
11	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1
12	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
13	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
14	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Another property of the Application object is `Select ion`, which returns the current selected object. If the current selected object is a cell, then a `Range` object is returned.

If you want to access a whole column, you can use the `Columns` property. Similarly, for a whole row, use the `Rows` property. These two properties return a `Range` object, so the followings are all equivalent to the examples in the first half of this section, provided that the workbook and the worksheet are activated.

```
ActiveSheet.Columns(2).Rows(2)
ActiveSheet.Rows(2).Columns(2)
Columns(2).Rows(2)
Rows(2).Columns(2)
```

5-3-2: Calculation Property

The `Calculation` property returns or sets the calculation mode. The type of this property is `XlCalculation`.

`XlCalculation` can be one of these constants.

- `xlCalculationAutomatic`: Calculates all dependent formulas every time you make a change to a value, formula, or name. This is the default calculation setting.
- `xlCalculationManual`: Calculates open workbooks only when you click `Calc Now` on the `Calculation` tab in the Excel options menu (or press F9).
- `xlCalculationSemiAutomatic`: Calculates all dependent formulas except data tables.

5-3-3: DisplayAlerts Property

The `DisplayAlerts` property determines if Excel will display alerts while a macro is running. By changing the value to `False`, Excel will choose the default response for all alerts that would have been displayed. One exception is when using the `SaveAs` method for workbooks, the `OverWrite` alert has a default response of `Yes`, but Excel will use `No` as a response when `DisplayAlerts` is `False`.

Excel will reset the `DisplayAlerts` property to `True` when the macro completes.

5-3-4: ThisWorkbook Property

The `ThisWorkbook` property will return the workbook that contains the macro that is currently running. Notice that it is not the same as `ActiveWorkbook` because it refers to the currently active workbook but not the workbook that actually contains the code being executed.

5-3-5: Calculate Method

The `Calculate` method forces all open workbooks to recalculate all cells that contain new, changed, or volatile cells and their dependents. This is similar to pressing the F9 key.

To calculate	Follow this example
All open workbooks	<code>Application.Calculate</code> (or just <code>Calculate</code>)
A specific worksheet	<code>Worksheets(1).Calculate</code>
A specified range	<code>Worksheets(1).Rows(2).Calculate</code>

5-3-6: FindFile Method

The `FindFile` method displays the Open dialog box and opens the file selected by the user. It returns a `True` value if a file was successfully opened or a `False` if the user clicked the Cancel button.

5-3-7: InputBox method

The `InputBox` method is very similar to the `InputBox` function in Chapter 2. However, the `InputBox` method allows you to ask the user to input a range of cells.

The syntax for the method is:

expression. `InputBox(Prompt, Title, Default, Left, Top, HelpFile, _
HelpContextId, Type)`

- *expression*: Required. An expression that returns an Application object.
- *Prompt*: Required String. The message to be displayed in the dialog box. This can be a string, a number, a date, or a Boolean value (Microsoft Excel automatically coerces the value to a String before it's displayed).
- *Title*: Optional Variant. The title for the input box. If this argument is omitted, the default title is "Input."
- *Default*: Optional Variant. Specifies a value that will appear in the text box when the dialog box is initially displayed. If this argument is omitted, the text box is left empty. This value can be a Range object.
- *Left*: Optional Variant. Specifies an *x* position for the dialog box in relation to the upper-left corner of the screen, in points.
- *Top*: Optional Variant. Specifies a *y* position for the dialog box in relation to the upper-left corner of the screen, in points.
- *HelpFile*: Optional Variant. The name of the Help file for this input box. If the *HelpFile* and *HelpContextId* arguments are present, a Help button will appear in the dialog box.
- *HelpContextId*: Optional Variant. The context ID number of the Help topic in *HelpFile*.
- *Type*: Optional Variant. Specifies the return data type. If this argument is omitted, the dialog box returns text. Can be one or a sum of the following values.

Value	Meaning
0	A formula
1	A number
2	Text (a string)
4	A logical value (True or False)
8	A cell reference, as a Range object
16	An error value, such as #N/A
64	An array of values

You can use the sum of the allowable values for *Type*. For example, for an input box that can accept both text and numbers, set *Type* to 1 + 2.

Remarks:

The dialog box has an OK button and a Cancel button. If you choose the OK button, `InputBox` returns the value entered in the dialog box. If you click the Cancel button, `InputBox` returns `False`.

If *Type* is 0, `InputBox` returns the formula in the form of text— for example, "`=2*PI () /360`". If there are any references in the formula, they are returned as A1-style references. (Use `ConvertFormula` to convert between reference styles.)

If *Type* is 8, `InputBox` returns a `Range` object. You must use the `Set` statement to assign the result to a `Range` object, as shown in the following example.

```
Set myRange = Application.InputBox(prompt := "Sample", type := 8)
```

If you don't use the `Set` statement, the variable is set to the value in the range, rather than the `Range` object itself.

Let's try! Corner

Ask the user to input a range of cells. Set those cells value 0.

5-3-8: Intersect Method

The `Intersect` method returns a `Range` object that represents the rectangular intersection of two or more ranges.

expression. `Intersect (Arg1, Arg2, ...)`

- *expression*: Optional. An expression that returns an `Application` object.
- *Arg1, Arg2, ...*: Required `Range`. The intersecting ranges. At least two `Range` objects must be specified.

For example,

```
Dim isect As Range
Worksheets("Sheet1").Activate
Set isect = Intersect(Range("A1:B3"), Range("B2:C4"))
If isect Is Nothing Then
    MsgBox "Ranges do not intersect"
Else
    isect.Select
End If
```

The above code results that the cells "B2:B3" are selected.

5-3-9: OnKey Method

The OnKey method runs a specified procedure when a particular key or key combination is pressed.

expression. OnKey (*Key*, *Procedure*)

- *expression*: Required. An expression that returns an Application object.
- *Key*: Required String. A string indicating the key to be pressed.
- *Procedure*: Optional Variant. A string indicating the name of the procedure to be run. If *Procedure* is "" (empty text), nothing happens when *Key* is pressed. This form of OnKey changes the normal result of keystrokes in Microsoft Excel. If *Procedure* is omitted, *Key* reverts to its normal result in Microsoft Excel, and any special key assignments made with previous OnKey methods are cleared.

Remarks:

The *Key* argument can specify any single key combined with ALT, CTRL, or SHIFT, or any combination of these keys. Each key is represented by one or more characters, such as "a" for the character a, or "{ENTER}" for the ENTER key.

To specify characters that aren't displayed when you press the corresponding key (ENTER or TAB, for example), use the codes listed in the following table. Each code in the table represents one key on the keyboard.

Key	Code
BACKSPACE	{BACKSPACE} or {BS}
BREAK	{BREAK}
CAPS LOCK	{CAPSLOCK}
CLEAR	{CLEAR}
DELETE or DEL	{DELETE} or {DEL}
DOWN ARROW	{DOWN}
END	{END}
ENTER	~ (tilde)
ENTER (numeric keypad)	{ENTER}
ESC	{ESCAPE} or {ESC}
F1 through F15	{F1} through {F15}
HELP	{HELP}
HOME	{HOME}
INS	{INSERT}
LEFT ARROW	{LEFT}
NUM LOCK	{NUMLOCK}
PAGE DOWN	{PGDN}
PAGE UP	{PGUP}
RETURN	{RETURN}
RIGHT ARROW	{RIGHT}
SCROLL LOCK	{SCROLLLOCK}
TAB	{TAB}
UP ARROW	{UP}

You can also specify keys combined with SHIFT and/or CTRL and/or ALT. To specify a key combined with another key or keys, use the following table.

To combine keys with	Precede the key code by
SHIFT	+ (plus sign)
CTRL	^ (caret)
ALT	% (percent sign)

To assign a procedure to one of the special characters (+, ^, %, and so on), enclose the character in braces. For details, see the example.

Example:

This example assigns "InsertProc" to the key sequence CTRL+PLUS SIGN and assigns "SpecialPrintProc" to the key sequence SHIFT+CTRL+RIGHT ARROW.

```
Application.OnKey "^{+}", "InsertProc"  
Application.OnKey "+^{RIGHT}", "SpecialPrintProc"
```

This example returns SHIFT+CTRL+RIGHT ARROW to its normal meaning.

```
Application.OnKey "+^{RIGHT}"
```

This example disables the SHIFT+CTRL+RIGHT ARROW key sequence.

```
Application.OnKey "+^{RIGHT}", ""
```

5-3-10: Quit Method

It causes Excel to quit.

5-3-11: Union Method

Returns the union of two or more ranges.

```
expression.Union(Arg1, Arg2, ...)
```

- *expression*: Optional. An expression that returns an Application object.
- *Arg1*, *Arg2*, ...: Required Range. At least two Range objects must be specified.

For example,

```
Dim isect As Range  
Worksheets("Sheet1").Activate  
Set isect = Union(Range("A1:B3"), Range("B2:C4"))  
isect.Select
```

The selected cells are as follows:

	A	B	C
1			
2			
3			
4			

5-4: The Workbooks Collection

You have learnt that you can access the Workbooks collection through the Workbooks property of the Application object. You can use the Workbooks collection to create, open, or delete workbooks.

5-4-1: Count Property

The Count property returns the number of workbooks in the Workbooks Collection.

5-4-2: Item Property

The Item property returns a single worksheet from the Workbooks Collection. As it is the default property for a Collection object, you usually use the short form described in section 5-3-1. In essence, the followings are equivalent.

Workbooks (1)
Workbooks.Item(1)

5-4-3: Add Method

The Add method creates a new workbook. The new workbook becomes the active workbook. It returns a Workbook object.

5-4-4: Close Method

The Close method closes all workbooks.

5-4-5: Open Method

The Open method opens a workbook.

expression.Open(*FileName*, *UpdateLinks*, *ReadOnly*, *Format*, *Password*, *_*
WriteResPassword, *IgnoreReadOnlyRecommended*, *Origin*, *Delimiter*, *Editable*, *_*
Notify, *Converter*, *AddToMru*, *Local*, *CorruptLoad*)

- *expression*: Required. An expression that returns the Workbooks object.
- *FileName*: Required String. The file name of the workbook to be opened.
- *UpdateLinks*: Optional Variant. Specifies the way links in the file are updated. If this argument is omitted, the user is prompted to specify how links will be updated. Otherwise, this argument is one of the values listed in the following table.

Value	Meaning
0	Doesn't update any references
1	Updates external references but not remote references
2	Updates remote references but not external references
3	Updates both remote and external references

- *ReadOnly*: Optional Variant. True to open the workbook in read-only mode.
- *Format*: Optional Variant. If Microsoft Excel is opening a text file, this argument specifies the delimiter character, as shown in the following table. If this argument is omitted, the current delimiter is used.

Value	Delimiter
1	Tabs
2	Commas
3	Spaces
4	Semicolons
5	Nothing
6	Custom character (see the Delimiter argument)

- *Password*: Optional Variant. A string that contains the password required to open a protected workbook. If this argument is omitted and the workbook requires a password, the user is prompted for the password.

- *WriteResPassword* (Advanced): Optional Variant. A string that contains the password required to write to a write-reserved workbook. If this argument is omitted and the workbook requires a password, the user will be prompted for the password.
- *IgnoreReadOnlyRecommended* (Advanced): Optional Variant. True to have Microsoft Excel not display the read-only recommended message (if the workbook was saved with the Read-Only Recommended option).
- *Origin* (Advanced): Optional Variant. If the file is a text file, this argument indicates where it originated (so that code pages and Carriage Return/Line Feed (CR/LF) can be mapped correctly). Can be one of the following `XlPlatform` constants: `xlMacintosh`, `xlWindows`, or `xlMSDOS`. If this argument is omitted, the current operating system is used.
- *Delimiter*: Optional Variant. If the file is a text file and the `Format` argument is 6, this argument is a string that specifies the character to be used as the delimiter. For example, use `Chr(9)` for tabs, use `,` for commas, use `;` for semicolons, or use a custom character. Only the first character of the string is used.
- *Editable* (Advanced): Optional Variant. It deals with the editability of add-ins. The default value is `False`.
- *Notify* (Advanced): Optional Variant. If the file cannot be opened in read/write mode, this argument is `True` to add the file to the file notification list. If this argument is `False` or omitted, no notification is requested, and any attempts to open an unavailable file will fail.
- *Converter* (Advanced): Optional Variant. The index of the first file converter to try when opening the file.
- *AddToMru* (Advanced): Optional Variant. `True` to add this workbook to the list of recently used files. The default value is `False`.
- *Local* (Advanced): Optional Variant. `True` saves files against the language of Microsoft Excel (including control panel settings). `False` (default) saves files against the language of Visual Basic for Applications (VBA) (which is typically US English unless the VBA project where `Workbooks.Open` is run from is an old internationalized XL5/95 VBA project).
- *CorruptLoad* (Advanced): Optional Variant. Can be one of the following constants: `xlNormalLoad`, `xlRepairFile` and `xlExtractData`. The Default behavior if no value is specified is usually normal.

Let's try! Corner

- 1 Download a series of Hang Seng Index from the Yahoo! Hong Kong website.
- 2 Use the `GetOpenFilename` method of the `Application` object to ask the user to get the file name to open. (You will probably want to check the usage of the method in the help file.)
- 3 Open the file as a new workbook using the `Open` method of the `Workbooks` collection.