ಐಟೆಕ್ ಅನಾಲಿಟಿಕ್ ಸಲೂಷನ್ಸ್

iTech Analytic Solutions

No. 9, 1st Floor,
8th Main, 9th Cross,
SBM Colony, Brindavan Nagar,
Mathikere, Bangalore – 560 054

Email: itechanalytcisolutions@gmail.com
Website: www.itechanalytcisolutions.com
Mobile: 9902058793

Chapter 1

Introduction

What is VBA?

- VBA is an acronym for Visual Basic for Applications
- Visual Basic for Applications is a programming feature designed by Microsoft for use with their Microsoft Office software package
- Specifically, it was designed for primary usage with Excel
- VBA is a tool that can be used to create programs to be run in Excel
- VBA should not be confused with VB, which is standard Visual Basic

Why VBA?

Microsoft Excel 2010 is an extremely powerful tool that you can use to manipulate, analyze, and present data. Sometimes though, despite the rich set of features in the standard Excel user interface (UI), you might want to find an easier way to perform a mundane, repetitive task, or to perform some task that the UI does not seem to address. Fortunately, Office applications like Excel have Visual Basic for Applications (VBA), a programming language that gives you the ability to extend those applications.

VBA works by running macros, step-by-step procedures written in Visual Basic. Learning to program might seem intimidating, but with some patience and some examples such as the ones in this article, many users find that learning even a small amount of VBA code makes their work easier and gives them the ability to do things in Office that they did not think were possible. Once you have learned some VBA, it becomes much easier to learn a whole lot more—so the possibilities here are limitless.

By far the most common reason to use VBA in Excel is to automate repetitive tasks. For example, suppose that you have a few dozen workbooks, each of which has a few dozen worksheets, and each of those needs to have some changes made to it. The changes could be as simple as applying new formatting to some fixed range of cells or as complex as looking at some statistical characteristics of the data on each sheet, choosing the best type of chart to display data with those characteristics, and then creating and formatting the chart accordingly.

Either way, you would probably rather not have to perform those tasks manually, at least not more than a few times. Instead, you could automate the tasks by using VBA to write explicit instructions for Excel to follow.

VBA is not just for repetitive tasks though. You can also use VBA to build new capabilities into Excel (for example, you could develop new algorithms to analyze your data, then use the charting capabilities in Excel to display the results), and to perform tasks that integrate Excel with other Office applications such as Microsoft Access 2010. In fact, of all the Office applications, Excel is the one most used as something that resembles a general development platform. In addition to all the obvious tasks that involve lists and accounting, developers use Excel in a range of tasks from data visualization to software prototyping.

Despite all of the good reasons to use VBA in Excel 2010, it is important to remember that the best solution to a problem might not involve VBA at all. Excel has a large range of features even without VBA, so even a power user is unlikely to be familiar with them all. Before you settle on a VBA solution, search the Help and online resources thoroughly to make sure that there is not a simpler way

When to Use VBA?

There are three principal reasons to consider VBA programming in Office 2010.

Automation & Repetition

VBA is effective and efficient when it comes to repetitive solutions to formatting or correction problems. For example, have you ever changed the style of the paragraph at the top of each page in Word? Have you ever had to reformat multiple tables that were pasted from Excel into a Word document or an Outlook e-mail? Have you ever had to make the same change in multiple Outlook contacts?

If you have a change that you have to make more than ten or twenty times, it may be worth automating it with VBA. If it is a change that you have to do hundreds of times, it certainly is worth considering. Almost any formatting or editing change that you can do by hand, can be done in VBA.

Extensions to User Interaction

There are times when you want to encourage or compel users to interact with the Office 2010 application or document in a particular way that is not part of the standard application. For example, you might want to prompt users to take some particular action when they open, save, or print a document.

Interaction between Office 2010 Applications

Do you need to copy all of your contacts from Outlook 2010 to Word 2010 and then format them in some particular way? Or, do you need to move data from Excel 2010 to a set of PowerPoint 2010 slides? Sometimes simple copy and paste does not do what you want it to do, or it is too slow. You can use VBA programming to interact with the details of two or more Office 2010 applications at the same time and then modify the content in one application based on the content in another.

Doing Things another Way

VBA programming is a powerful solution, but it is not always the optimal approach. Sometimes it makes sense to use other ways to achieve your aims.

The critical question to ask is whether there is an easier way. Before you begin a VBA project, consider the built-in tools and standard functionalities. For example, if you have a time-consuming editing or layout task, consider using styles or accelerator keys to solve the problem. Can you perform the task once and then use CTRL+Y (Redo) to repeat it? Can you create a new document with the correct format or template, and then copy the content into that new document?

Office 2010 applications are powerful; the solution that you need may already be there. Take some time to learn more about Office 2010 before you jump into programming.

Before you begin a VBA project, ensure that you have the time to work with VBA. Programming requires focus and can be unpredictable. Especially as a beginner, never turn to programming unless you have time to work carefully. Trying to write a "quick script" to solve a problem when a deadline looms can result in a very stressful situation. If you are in a rush, you might want to use conventional methods, even if they are monotonous and repetitive

History of VBA

The incredible popularity of Visual Basic shortly after its launch prompted Microsoft to wonder if a "cut down" version of the product could replace the many different macro languages lurking behind its range of business applications. Bill Gates talked for many years—since the days of DOS—of a universal batch language. This goal is now coming to fruition in the shape of VBA. However, as the following chronology shows, this goal wasn't achieved overnight:

1993—VBA launched with Microsoft Excel

VBA first saw the light of day as a replacement for the macro language in Microsoft Excel. Its power and sophistication in comparison to the original macro languages made it an instant success with those developers creating custom solutions with Excel.

1994—VBA included with Microsoft Project

Perhaps because Microsoft Project had to be customized in many situations to satisfy the wide and varied needs of project managers, Project was next on the list of applications to be VBA-enabled.

1995—Included with Microsoft Access, replacing Access Basic

Perhaps the biggest boost to VBA came when Access Basic (a subset of VBA written specifically for Access) was replaced with the full version of VBA. Many of today's VB programmers apprenticed on VBA in Access, cutting their teeth on custom applications using VBA and Access. Many Access developers have moved on to the full version of Visual Basic to create full three-tier client server applications.

1996—VBA becomes the language element of Visual Basic

1996 saw the launch of Visual Basic 4.0, a massive leap forward and almost a totally different product from VB3. Written in C++, Version 4 was a ground-up rewrite of VB, whose previous versions were written in assembler. With VB4, VB became object-oriented; VB could be used to create class models and DLLs, as well as to easily reference external object libraries. Part of the componentization of Visual Basic was the use of a separate language library, VBA. Some intrinsic language elements remained in the VB and VB runtime libraries for backward compatibility, but most were transferred to the VBA library, and many were completely rewritten.

1996—Included with Word, replacing Word Basic

Many people were surprised that Word Basic was the last of the Microsoft macro languages to hit the dust. This appears to have happened partly because the demand for customized word processor applications is much smaller than for customized applications using the other components of the Office suite, and because the core of Word developers were initially opposed to a change to VBA.

1997—VBA 5.0 launched, covering the complete range of Office 97 products

With the inclusion of VBA in PowerPoint, all the members of Office 97 (with the exception of Outlook, which is VBScript-enabled) now include VBA as their programming language.

1997—Microsoft licenses VBA for use with other software

Over 50 major software vendors licensed VBA within the first few weeks of Microsoft's announcement. The fact that so many leading companies have chosen to license VBA bodes well for the future. In learning VBA now, you are building a skill set that will be in demand for a long time to come.

1998—VB and VBA Version 6 launched

The language continues to expand, although not at the same rate as previously. Interestingly, with the exception of two functions, the new functions in VBA have come from VBScript. The rest of the new functionality available to VB/VBA developers comes in the form of several new object models, which is likely to be the way VB and VBA will expand in the future

Advantages of VBA

The large portion of work in MS Excel can be achieved via macro and VBA. This post is an attempt to help MS Excel user with VBA. The post below provides an overview about Macro/VBA

- You can automate almost anything you do in Excel. To do so, you will have to write
- Instructions that you want Excel to perform. Automating a task by using VBA offers several advantages:
- Excel always executes the task in exactly the same way and with more consistency.
- VBA performs the task much faster than you can do it manually.
- If you're a good macro programmer, Excel always performs the task without errors.
- If you set things up properly, someone who doesn't know anything about Excel can perform the task.
- With MS Excel you can do things which are otherwise looks impossible — Also, it makes you popular in your work place
- For long, time-consuming and monotonous tasks, Excel performs the entire task while you enjoy leisure

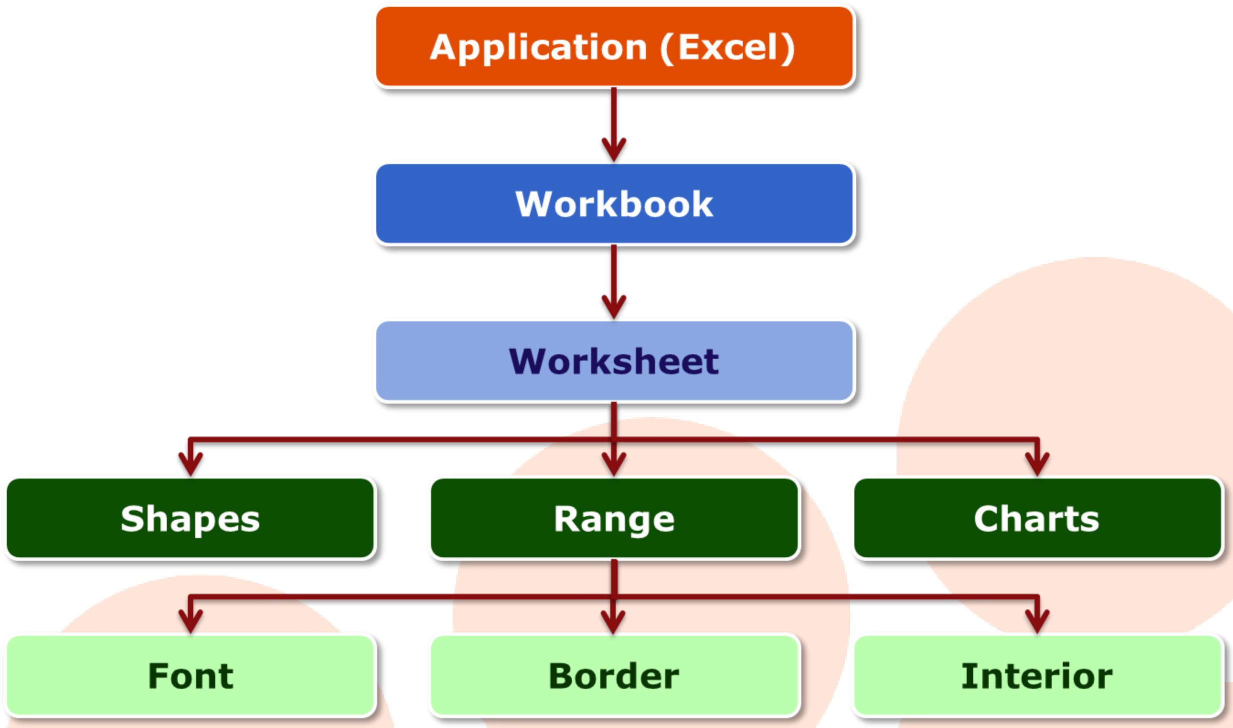
Under MS Excel help, VBA is little unexplored by most of the users. Everyone knows that's things are possible using VBA. But, we look for work around.

Disadvantages of VBA

- You have to learn how to write programs in VBA. Fortunately, it's not as difficult as you might expect
- Other people who need to use your VBA programs must have their own copies of Excel. It would be nice if you could press a button that transforms your Excel/VBA application into a stand-alone program, but that isn't possible (and probably never will be)
- Sometimes, things go wrong. In other words, you can't blindly assume that your VBA program will always work correctly under all circumstances. Welcome to the world of debugging
- VBA is a moving target. As you know, Microsoft is continually upgrading Excel. You may discover that VBA code you've written doesn't work properly with a future version of Excel

Understanding Object Hierarchy

Most applications consist of many objects arranged in a hierarchy



Objects, Methods, Properties and Variables

Each line of code generally has the same structure (which is also known as **Syntax**). VBA is loosely based around the concept of **Object Orientated Programming** (OOP) and the following syntax is used to define how you write code using any of the libraries that are loaded.

OBJECT.*Identifier*[.*sub_Identifier*]

The square brackets wrapped around the *sub_Identifier* is the convention meaning it is optional and therefore not always required.

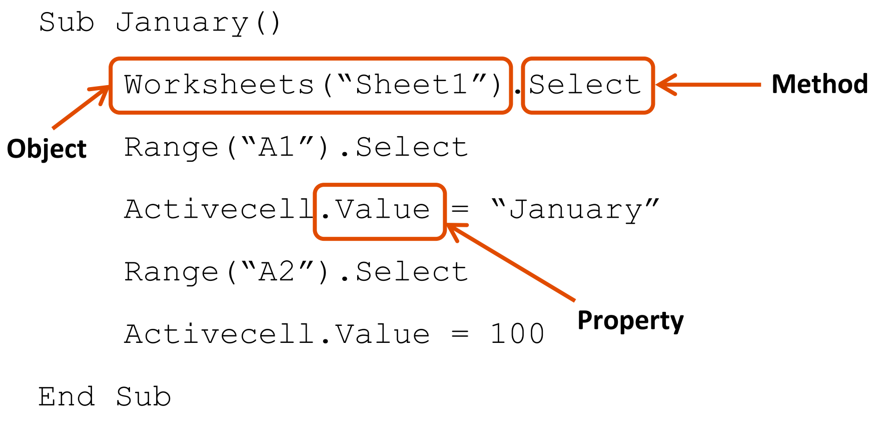
An *Identifier* and *sub_Identifier* can be one of three types:

1. Property
2. Method
3. Event

Similar to physical objects such as a car or a chair, the application objects, as listed above, have **Properties** and **Methods** (*as well as **Events***)

Object	Property	Method
Car	Color	Accelerate
ActiveCell	Value	
Worksheets("Sheet1")		Select

Identifying the Objects, Methods and Properties from the previous example.



```
Sub January()
      Worksheets("Sheet1").Select
      Range("A1").Select
      Activecell.Value = "January"
      Range("A2").Select
      Activecell.Value = 100
End Sub
```

Examples

1. Create a new blank workbook.
2. Click on the **Tools** menu, select **Macro** and choose Visual Basic Editor.
3. Click on the **Insert** menu and select **Module**.

Properties

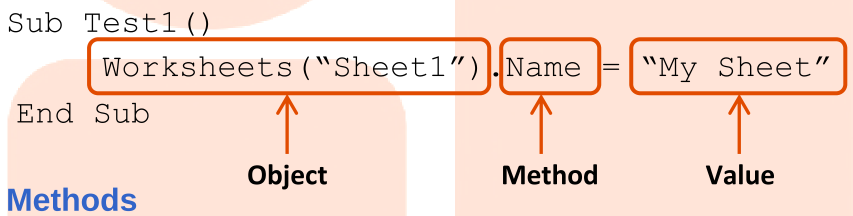
A **Property** is an attribute of an object, e.g. the colour of a car, the name of a worksheet.

Object.Property = Value

Car.Colour = Red

Worksheets("Sheet1").Name = "My Sheet"

The following example will change the name of "Sheet1" to "My Sheet".



```
Sub Test1()
      Worksheets("Sheet1").Name = "My Sheet"
End Sub
```

Methods

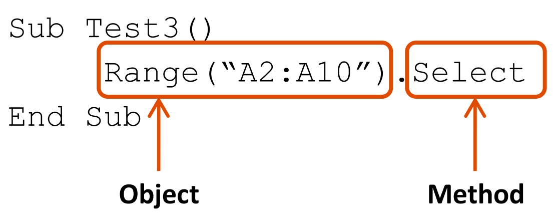
A Method is an activity that an object can be told to do, e.g. accelerate the car, select a cell, insert a worksheet, delete a worksheet.

Object.Method

Car.Accelerate

Range("A2:A10").Select

The following example will select a range of cells (A2 to A10) in the current worksheet.



```
Sub Test3()
     Range("A2:A10").Select
End Sub
```

Methods that contain arguments

There are methods that contain many arguments, for example inserting a worksheet(s). The numerous arguments contain information about how many worksheets you would like to insert, the position of the worksheet(s) and the type of the worksheet(s).

Object.Method Argument1,Argument2,...

Example:

Worksheets.Add Before, After, Count, Type

Add	Add a new worksheet.
Before/After	Before which worksheet? After which worksheet?
Count	How many worksheets
Type	What type of worksheet ie worksheet, chart sheet etc

The following example will place 2 new sheets after Sheet 2.

Worksheets.Add, Sheets("Sheet2"), 2

- The comma after Add represents the "Before" argument.
- If "Type" is omitted, then it will assume the Default Type. The Default Type is xlworksheet.

```
 Sub Insert2Sheets()
      Worksheets.Add Sheets("Sheet2"), 2
End Sub
```

Events

Events are like Methods the only difference being when and who/what calls this action. It is an action like a method but the system will trigger it for you instead of the user invoking the action.

This is very useful when a system changes state and needs to automate a procedure on behalf of the user.

In fact, there are many events being triggered all the time; users are simply not aware of this unless there is a procedure to trigger it. The system constantly listens for the event to take place.

When you use standard Excel features like **Data Validation** or **Conditional Formatting**, the system automatically creates code for an event so when users enter a value in a cell it will automatically trigger the feature and validate and format the active cell without a user calling the a macro manually. This type of event is normally known as '**Enter**' for a Worksheet.

There are many predefines events which have no code just a starting and ending signature and users need to add code in between these signatures.