

ಐಟಿಕ್ ಅನಾಲಿಟಿಕ್ ಸಲೂಷನ್ಸ್

iTech Analytic Solutions

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

Email: itechanalytcsolutions@gmail.com

Website: www.itechanalytcsolutions.com

Mobile: 9902058793



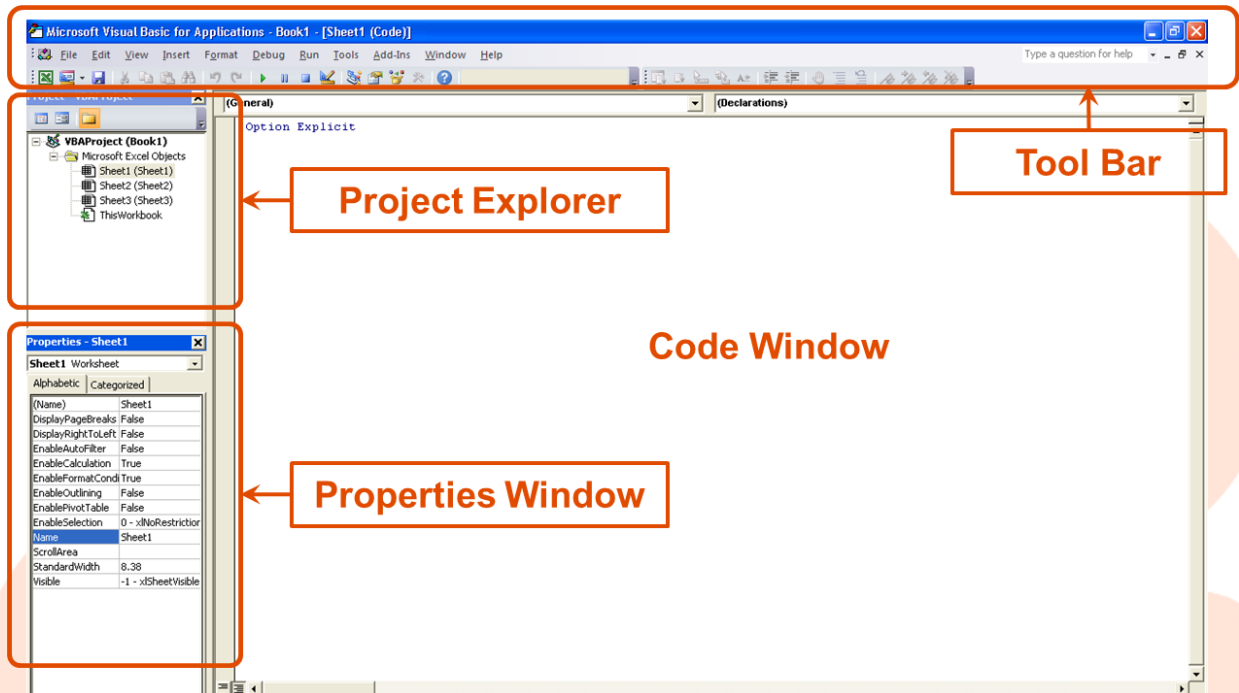
Chapter 3

The VBA Editor

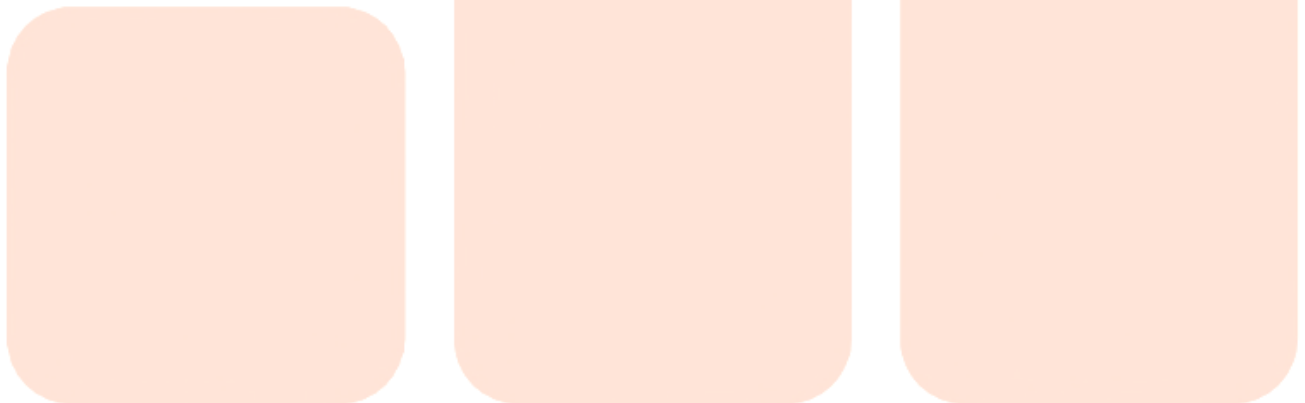


The VBA Editor Screen

The easiest way to access the Visual Basic Editor in Excel is to press the key combination ALT-F11 (ie. press the ALT key, and while this is pressed down, press F11). You will be presented with the Visual Basic Editor, similar to the image below (note that your normal Excel window will remain open behind this window).

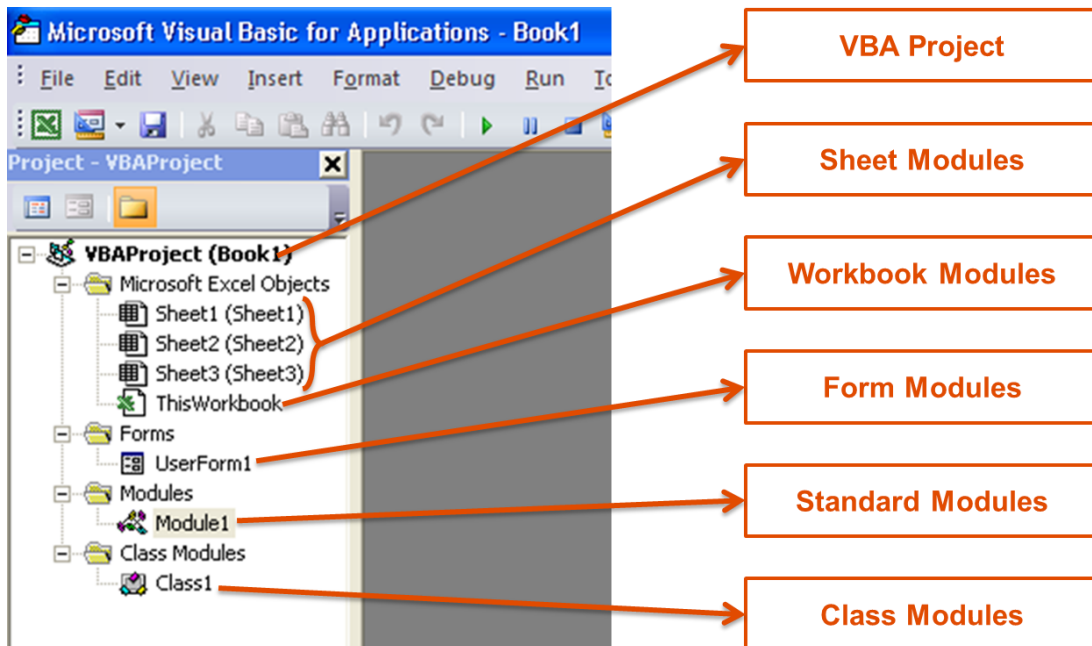


There are a number of windows that can be viewed in the Excel VBA Editor. These are controlled via the **View** menu at the top of the VBA Editor window. The individual windows are described below.





The Project Explorer



The **Project Explorer** opens up on the left of the VBA Editor (as shown in the above image). This window provides you with an index of all Visual Basic files, linked to all open Excel workbooks. These consist of :

A file (or a **worksheet object**), linked to each worksheet of your Excel workbook and Additional files (called **Modules**), which contain user-defined code and are not linked to any specific worksheet

If you want to write a macro that is to be linked to a specific worksheet, then this code should be entered into the relevant worksheet object. If your code is more general, you may want to create a new module to hold this.

To show the project explorer click the Project Explorer icon or Press CTRL + R on the keyboard. The Project Explorer allows you to "explore" the VBA code for all Excel workbooks you have open. VBA codes are written in "modules" in the Project Explorer.

There are 5 different types of modules:

1. Standard Modules

- Also called simply Code Modules or just Modules
- Most of the VBA codes are written here
- Basic macros and Custom function (User Defined Functions) should be in these modules only
- A workbook's VBA Project can contain as many standard code modules . This makes it easy to split the procedure into different modules for organization and ease of maintenance

For example, put all the database procedures in a module named DataBase, and all the mathematical procedures in another module called Math. As long as a procedure isn't declared with the Private keyword, or the module isn't marked as private, any procedures can be called in any module from any other module without doing anything special. .

2. Worksheet modules

- By Default there will be a module for each worksheet in the workbook



- If there is a code that is specific to a worksheet VBA code can be written in the worksheet module for that worksheet
- But it is not recommended. Instead insert a module in a module folder and write your code there

3. Workbook module

- By default there will be a Workbook module in the Project

4. Form modules

- Forms can be created by creating Form Modules
- Codes are written for each objects (controls) in the Forms

5. Class modules

- Used in Advanced VBA Programming
- Used to create objects and object Models
- Used to trap events
- Used to raise events
- Used to create your own objects and object models

The Code Windows

In the Visual Basic Editor image above, the code window for the Worksheet, Sheet1, is displayed (although it contains no VBA code at present).

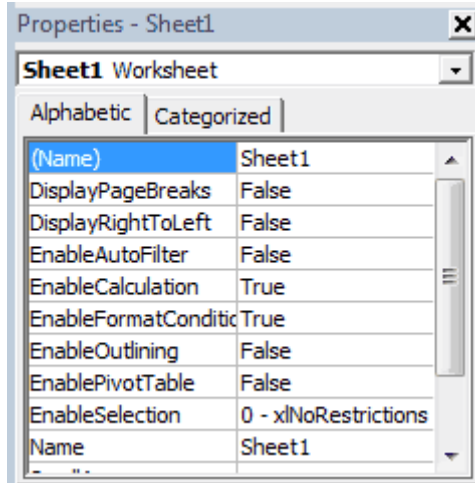
The **Code Window** contains the VBA code for each individual worksheet object or module. Once a worksheet object or module is opened up, you can type your VBA code directly into the code window and the VBA editor assists you with typing in valid VBA code by highlighting lines that do not adhere to the VBA code rules, as you type.

The code window for an existing worksheet object or module is opened up by double-clicking on the worksheet or module name in the Project Window. Alternatively, you can create a new module as follows:

1. If you have more than one Excel workbook open, in the 'Project Window', select the workbook you want to add a module to
2. Right click in the 'Project Window' and select **Insert->Module**



The Properties Window



The **Properties Window** lists the properties of the object that is selected in the *Project Window* at *design time* (ie. not during *run time*). These properties vary, depending on the type of object that is selected (a worksheet, workbook, module, etc).

The Immediate Window

View the **Immediate Window** by selecting **View→Immediate Window** from the Visual Basic Editor, or by pressing **CTRL-G**. This window assists with the debugging of code by allowing you to execute individual lines of code. This is done by typing in an individual line code and then pressing enter to execute it.

The Locals Window

View the **Locals Window** by selecting **View→Locals Window** from the Visual Basic Editor. This window displays all local variables that are declared in the current procedure. It is split into columns which show the name, value and type of each variable and updates these values automatically, as the programme is executed. The Locals window is therefore useful for debugging VBA code.

The Watch Window

The **Watch Window** is also useful when debugging VBA code, as it shows the value, type and context of any watch expressions that have been defined by the user.

The Watch Window can be opened by selecting **View→Watch Window** from the Visual Basic Editor, or the window appears automatically when a 'watch' expression is defined. To define a 'watch' expression:

1. Highlight an expression within the VBA code
2. From the **Debug** menu at the top of the VBA editor, select the option **Quick Watch...**
3. Click on **Add**

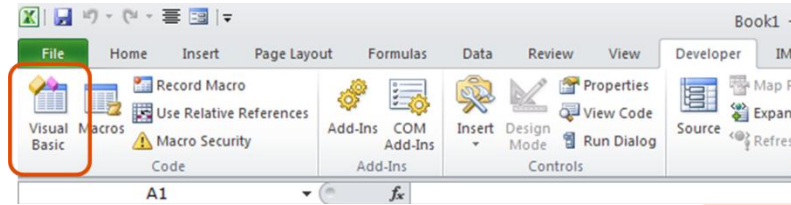
The menus at the top of the Excel Visual Basic Editor contain numerous options and commands for designing, executing and debugging VBA code



Opening the VBA Editor

Following are different ways to open the VBA Editor

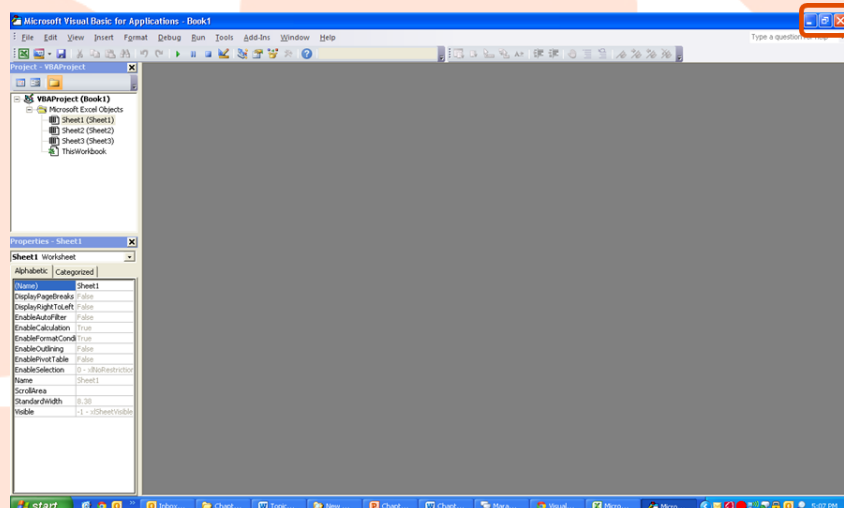
- Press Alt + F11 to open the VBA Editor
- Click on Developer Tab and Click on Visual Basic command



Closing the VBA Editor

Following are different ways to close the VBA Editor

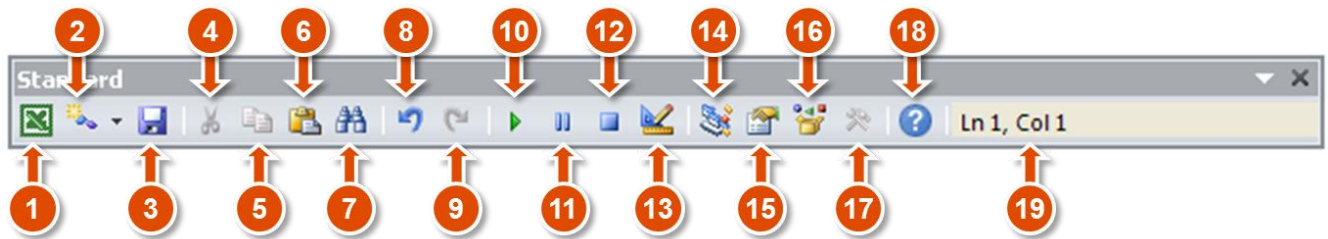
- Press Alt + F4 to close the VBA Editor
- Click on the “X” button on the Top right of the screen





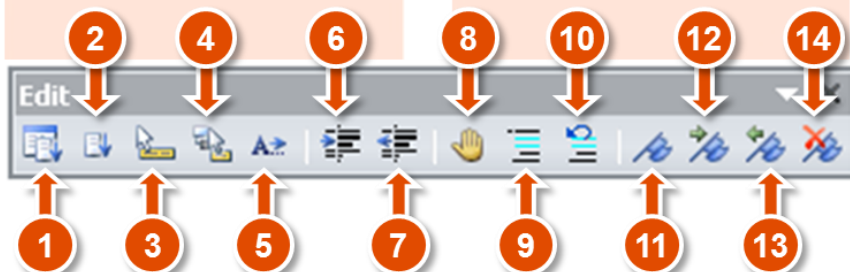
Working with Toolbars

Standard Toolbar



1. Takes you to Excel Window (Alt + F11)
2. Inserts a Procedure or Function
3. Saves the Workbook (Ctrl + S)
4. Cut's the selected Text (Ctrl + X)
5. Copy's the selected Text (Ctrl + C)
6. Paste's the copied text (Ctrl + V)
7. Find the text looking for (Ctrl + F)
8. Undo the previous action (Ctrl + Z)
9. Redo the previous action (Ctrl + Y)
10. Execute / Run the Procedure / UserForm (F5)
11. Stop / Break the Procedure executing (Ctrl + Break)
12. Reset the Executing Procedure
13. Go to Design Mode
14. Enable Project Explorer (Ctrl + R)
15. Enable Properties Window (F4)
16. Enable Object Browser (F2)
17. Enable Toolbox
18. Enable VBA Help
19. Show which line and column the cursor is placed in the Code

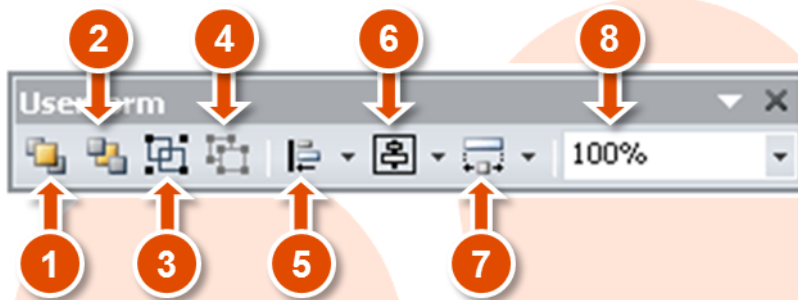
Edit Tool Bar



1. List Properties / Methods (Ctrl + J)
2. List Constants (Ctrl + Shift + J)
3. Quick Information about the selected parameters, arguments, etc. (Ctrl + I)
4. Parameter Information (Ctrl + Shift + I)

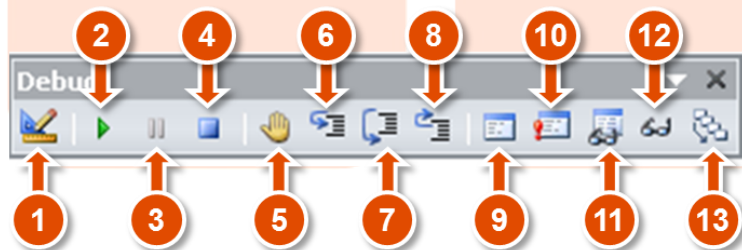
5. Complete the typed word (Ctrl + Space)
6. Indent the code (Tab)
7. Outdent the code (Shift + Tab)
8. Add / Remove Breakpoint (F9)
9. Comment the Block
10. Uncomment the Block
11. Add / Remove Bookmark
12. Go to Next Bookmark
13. Go to Previous Bookmark
14. Clear all the Bookmarks added

UserForm Toolbar



1. Bring the selected Control to Front (Ctrl + J)
2. Move the selected Control to Back (Ctrl + K)
3. Group the selected Controls
4. Ungroup the selected Controls
5. Align the selected Controls (Left, Center, Right, Top, Middle, Bottom, To grid)
6. Align the selected Control Center of the form (Horizontally, Vertically)
7. Make the selected Control same width, height or both
8. Zoom the form

Debug Toolbar



1. Design Mode
2. Execute / Run the Procedure / UserForm (F5)
3. Stop / Break the Procedure executing (Ctrl + Break)
4. Reset the Executing Procedure
5. Add / Remove Breakpoint (F9)



6. Step Into (F8)
7. Step Over (Shift + F8)
8. Step Out (Ctrl + Shift + F8)
9. Enable Local Window
10. Enable Immediate Window (Ctrl + G)
11. Enable Watch Window
12. Enable Quick Watch Window
13. Enable Call Stack (Ctrl + L)

Working with a Code Module

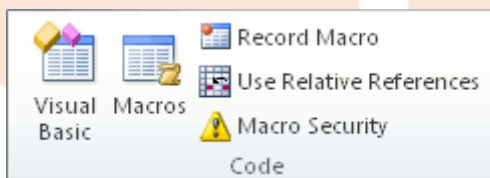
- VBA code is typed in the VBA Editor in what are called modules
- A VBA module resembles a Word document in both organization and typing
- The commands in the modules can be executed to control Microsoft Excel
- The VBA modules themselves are organized in what is called a VBA project
- A VBA project is defined as a collection of VBA modules and other programming elements
- When a new workbook file is created in Microsoft Excel, a new VBA project is automatically created and associated with that workbook
- A workbook can contain only 1 VBA project
- VBA modules, code and other elements can be added to a VBA project when needed
- Macros are executed from the workbook they were created in
- The workbook should be opened to execute the code

Different ways of Running the Macro created

There are several ways to run a macro in Microsoft Excel. A macro is an action or a set of actions that you can use to automate tasks.

Run the macro

1. Open the workbook that contains the macro.
2. On the **Developer** tab, in the **Code** group, click **Macros**.
3. In the **Macro name** box, click the macro that you want to run.



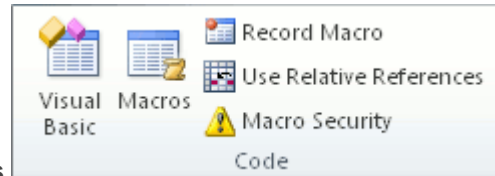
4. Do one of the following:
 - To run a macro in an Excel workbook, click **Run**.
 - To run a macro from a Microsoft Visual Basic for Applications (VBA) module, click **Edit**, and then on the **Run** menu, click **Run Sub/UserForm** , or press F5.



Tip: You can also press CTRL+F8 to run the macro. You can interrupt the execution of the macro by pressing ESC

Run a macro by pressing a CTRL combination shortcut key

1. If the **Developer** tab is not available, do the following to display it:
 - Click the **File** tab, click **Options**, and then click the **Customize Ribbon** category.
 - In the **Main Tabs** list, select the **Developer** check box, and then click **OK**.



2. On the **Developer** tab, in the **Code** group, click **Macros**.
3. In the **Macro name** box, click the macro that you want to assign to a CTRL combination shortcut key.
4. Click **Options**.
The **Macro Options** dialog box appears.
5. In the **Shortcut key** box, type any lowercase letter or uppercase letter that you want to use with the CTRL key.

Note The shortcut key will override any equivalent default Excel shortcut key while the workbook that contains the macro is open.

6. In the **Description** box, type a description of the macro.
7. Click **OK** to save your changes, and then click **Cancel** to close the **Macro** dialog box.

Run a macro by clicking a button on the Quick Access Toolbar

To add a button to the Quick Access Toolbar that will run a macro, do the following:

1. Click the **File** tab, **Options**, and then click **Quick Access Toolbar**.
2. In the **Choose commands from** list, select **Macros**.
3. In the list, click the macro that you created, and then click **Add**.
4. To change the button image of the macro, select the macro in the box to which it was added, and then click **Modify**.
5. Under **Symbol**, click the button image that you want to use.
6. To change the name of the macro that is displayed when you rest the pointer on the button, in the **Display name** box, type the name that you want to use.
7. Click **OK** to add the macro button to the **Quick Access Toolbar**.
8. On the **Quick Access Toolbar**, click the macro button that you just added

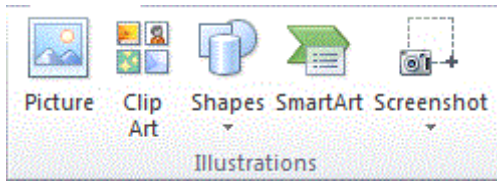
Run a macro by clicking an area on a graphic object


You can create a hot spot on a graphic that users can click to run a macro.

1. In the worksheet, insert a graphic object, such as a picture, clip art, shape, or SmartArt.
To learn about inserting a graphic object, see Add, change, or delete shapes.



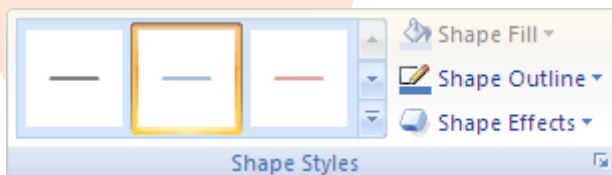
- To create a hot spot on the existing object, on the **Insert** tab, in the **Illustrations** group, click **Shapes**, select the shape that you want to use, and then draw that shape on the existing object.



- Right-click the hot spot that you created, and then click **Assign Macro**.
- Do one of the following:
 - To assign an existing macro to the graphic object, double-click the macro or enter its name in the **Macro name** box.
 - To record a new macro to assign to the selected graphic object, click **Record**, type a name for the macro in the **Record Macro** dialog box, and then click **OK** to begin recording your macro. When you finish recording the macro, click **Stop Recording**  on the **Developer** tab in the **Code** group.

Tip You can also click **Stop Recording**  on the left side of the status bar.

- To edit an existing macro, click the name of the macro in the **Macro name** box, and then click **Edit**.
- Click **OK**.
 - In the worksheet, select the hot spot. This displays the **Drawing Tools**, adding a **Format** tab.
 - On the **Format** tab, in the **Shape Styles** group, click the arrow next to **Shape Fill**, and then click **No Fill**.



- Click the arrow next to **Shape Outline**, and then click **No Outline**

Configure a macro to run automatically upon opening a workbook

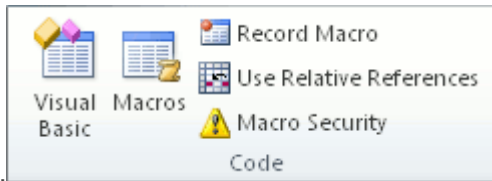
If you record a macro and save it with the name "Auto_Open," the macro will run whenever you open the workbook that contains the macro. Another way to automatically run a macro when you open a workbook is to write a VBA procedure in the **Open** event of the workbook by using the [Visual Basic Editor](#). The **Open** event is a built-in workbook event that runs its macro code every time you open the workbook.

Create an Auto_Open macro

- If the **Developer** tab is not available, do the following to display it:
 - Click the **File** tab, and then click **Options**.
 - In the **Customize Ribbon** category, in the **Main Tabs** list, select the **Developer** check box, and then click **OK**.
- To set the security level temporarily to enable all macros, do the following:



- On the **Developer** tab, in the **Code** group, click **Macro Security**




- In the **Macro Settings** category, under **Macro Settings**, click **Enable all macros (not recommended, potentially dangerous code can run)**, and then click **OK**.

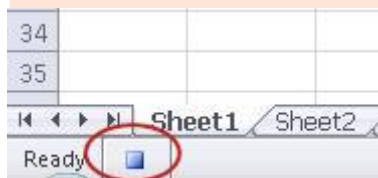
Note To help prevent potentially dangerous code from running, we recommend that you return to any one of the settings that disable all macros after you finish working with macros.

1. If you want to save the macro with a particular workbook, open that workbook first.
2. On the **Developer** tab, in the **Code** group, click **Record Macro**.
3. In the **Macro name** box, type **Auto_Open**.
4. In the **Store macro in** list, select the workbook where you want to store the macro.

Tip If you want a macro to be available whenever you use Excel, select **Personal Macro Workbook**. When you select **Personal Macro Workbook**, Excel creates a hidden personal macro workbook (Personal.xlsm), if it does not already exist, and saves the macro in this workbook. In Windows Vista, this workbook is saved in the C:\Users*user name*\AppData\Local\Microsoft\Excel\XLStart folder. If you can't find it there, it may have been saved in the Roaming subfolder instead of Local. In Microsoft Windows XP, this workbook is saved in the C:\Documents and Settings*user name*\Application Data\Microsoft\Excel\XLStart folder. Workbooks in the XLStart folder are opened automatically whenever Excel starts. If you want a macro in the personal macro workbook to be run automatically in another workbook, you must also save that workbook in the XLStart folder so that both workbooks are opened when Excel starts.

5. Click **OK**, and then perform the actions that you want to record.
6. On the **Developer** tab, in the **Code** group, click **Stop Recording** .

Tip You can also click **Stop Recording** on the left side of the status bar.



Notes

- If you chose to save the macro in **This Workbook** or **New Workbook** in step 6, save or move the workbook into one of the XLStart folders.
- Recording an Auto_Open macro has the following limitations:
 - If the workbook where you save the Auto_Open macro already contains a VBA procedure in its **Open** event, the VBA procedure for the **Open** event will override all actions in the Auto_Open macro.
 - An Auto_Open macro is ignored when a workbook is opened programmatically by using the **Open** method.



- An Auto_Open macro runs before any other workbooks open. Therefore, if you record actions that you want Excel to perform on the default Book1 workbook or on a workbook that is loaded from the XLStart folder, the Auto_Open macro will fail when you restart Excel, because the macro runs before the default and startup workbooks open.

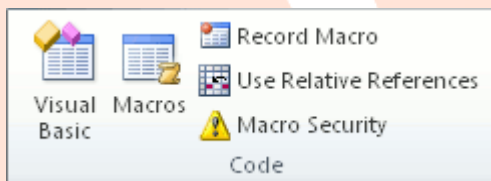
If you encounter these limitations, instead of recording an Auto_Open macro, you must create a VBA procedure for the **Open** event as described in the next section of this article.

- If you want Excel to start without running an Auto_Open macro, hold down the SHIFT key when you start Excel.

Create a VBA procedure for the Open event of a workbook

The following example uses the **Open** event to run a macro when you open the workbook.

1. If the **Developer** tab is not available, do the following to display it:
 - Click the **File** tab, and then click **Options**.
 - In the **Customize Ribbon** category, in the **Main Tabs** list, select the **Developer** check box, and then click **OK**.
2. To set the security level temporarily to enable all macros, do the following:
 - On the **Developer** tab, in the **Code** group, click **Macro Security**.



- In the **Macro Settings** category, under **Macro Settings**, click **Enable all macros (not recommended, potentially dangerous code can run)**, and then click **OK**.

Note To help prevent potentially dangerous code from running, we recommend that you return to any one of the settings that disable all macros after you finish working with macros.

1. Save and close all open workbooks.
2. Open the workbook where you want to add the macro, or create a new workbook.
3. On the **Developer** tab, in the **Code** group, click **Visual Basic**.
4. In the Project Explorer window, right-click the **ThisWorkbook** object, and then click **View Code**.

Tip If the Project Explorer window is not visible, on the **View** menu, click **Project Explorer**.

5. In the **Object** list above the Code window, select **Workbook**.

This automatically creates an empty procedure for the **Open** event, such as this:

```
Private Sub Workbook_Open()
```

```
End Sub
```



6. Add the following lines of code to the procedure:

```
Private Sub Workbook_Open()  
    MsgBox Date  
    Worksheets("Sheet1").Range("A1").Value = Date  
End Sub
```

7. Switch to Excel and save the workbook as a macro-enabled workbook (.xlsm).
8. Close and reopen the workbook. When you open the file again, Excel runs the Workbook_Open procedure, which displays today's date in a message box.
9. Click **OK** in the message box.

Note that cell A1 on Sheet1 also contains the date as a result of running the Workbook_Open procedure

Setting Breakpoints in the Editor

You can set a breakpoint to suspend a macro that is running at a specific statement in the macro. Typically, you set a breakpoint where you suspect a problem exists. You clear breakpoints when you no longer need them to stop the macro

Setting Breakpoints

To set a breakpoint, use either of the following methods

Method 1:

1. Click anywhere in the line of code at which you want the macro to halt.
A flashing pointer appears
2. On the Debug menu, click Toggle Breakpoint (or press F9)
A breakpoint is defined for that line of code (provided that a breakpoint was not previously defined for the line of code).

Method 2:

1. Right-click the line of code that contains the breakpoint.
2. On the shortcut menu, point to Toggle, and then click Breakpoint.

When you run this code, the macro halts on the first line of code that contains a breakpoint. Note that you can use more than one breakpoint in a macro.

Clearing Breakpoints

1. Click anywhere in the line of code that contains the breakpoint you want to remove.
2. On the Debug menu, click Toggle Breakpoint (or press F9). The breakpoint is removed from the line of code.

NOTE: Breakpoints are not saved with your code. So, you must configure breakpoints for every debugging session

Using Break Mode

When your code pauses, for example, when it encounters a breakpoint, the macro is in break mode. Break mode allows you to view the current condition of the macro. When a macro is in break mode, you can look at the values of your variables. You can also step through the code one line at a time to trace the logic of the macro.



Entering Break Mode

One method for entering break mode involves stepping through code from the beginning of the macro. To do this, follow these steps:

1. Click the line of code that contains the first instruction of the macro
2. On the Debug menu, click Step Into.
An arrow appears to the left of the first line of code.
3. On the Debug menu, click Step Over
This step advances the arrow to the next instruction. If the instruction that is highlighted is a procedure (function, sub, or property), using the Step Over command executes the procedure as a unit without stepping through the code

To step through a procedure line by line, click Step Into

NOTE: The Debug menu commands are also available on the Debug toolbar.

Exiting Break Mode

To exit break mode, click Reset on the Run menu. This ends the break mode session.

Using ToolTips

When you use break mode, ToolTips indicate the current value of a specified variable

To see a sample value displayed in a Tooltip, follow these steps:

1. In a Visual Basic module, type the following sample subroutine:

```
Sub Test()  
    Dim Name As String  
    Name = "Kerry"  
    Name = "Nancy"  
End Sub
```

2. On the Debug menu, click Step Into (or press F8) to step through the subroutine
Click Step Into again until the Name = "Kerry" line is highlighted, and then move the pointer over the Name variable
A Tooltip appears with the following text:
Name = ""
3. Press F8 to execute the line of code that assigns the Name variable
The Name = "Nancy" line should be highlighted
4. Move the pointer over the Name variable. A Tooltip appears with the following text:
Name = "Kerry"

Stepping Through Code



One of the first methods to debug code is to step through the code one line at a time. To step through code, put the cursor on the first line of code to be analyzed and press **F8** or choose *Step Into* on the Debug menu. The next line of code to be executed will be displayed in yellow background with a black font.

Note that the highlighted line of code has not yet been executed -- it is the next line to execute.

If your code calls another procedure, stepping through the code with F8 will cause execution to enter the called procedure in a line-by-line sequence. If you want to execute the called procedure without stepping through it, press **SHIFT F8**. This will execute the called procedure and then pause on the line of code after calling the procedure. If you are already stepping through a procedure, you can press **CTRL F8** to resume code execution line-by-line. At any time you are paused either in step-by-step mode or at a breakpoint (see below), you can press **F5** or *Continue* from the Run menu to cause VBA to run to completion or until a pause statement is encountered.

Whenever you are paused in step-by-step mode, you can query or change a variable's value from the immediate window

