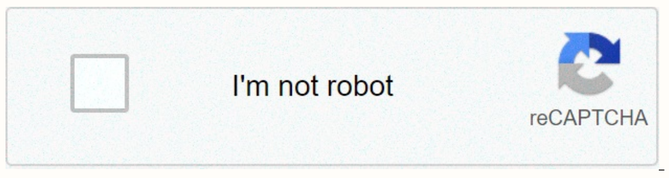

Continue

Save workbook excel template

If you've already worked with Excel, you've probably been fully familiar with 2 basic commands to save workbooks: it may surprise you to know that when working with VBA, you can perform the same activities. Actually, knowing how to save Excel workbook with VBA is essential. As you work with Visual Basic for Applications, you will notice that saving the workbook is one of the most important things your macros can do. Considering the importance of knowing how to save workbook using VBA, this Excel tutorial focuses on this specific subject: How to save Excel workbook with VBA. Besides providing a few instances of VBA code that you can use to save the workbook, I explain the basics around 4 VBA methods that you are likely to encounter and use continuously while saving the workbook. The following table of contents shows the specific topics that I will explain in this Excel tutorial: This Excel tutorial does not cover the subject of saving Excel workbook as PDF using VBA. I explain how to export Excel files to PDF using macros, and provide a few code examples, here. Let's start looking at the basic ways to save Excel workbook using VBA. How to save Excel workbook with workbook.Save VBA is the most basic method to save Excel workbook with VBA workbook.Save method. Saves the corresponding workbook. In other words, the Workbook.Save method is roughly equivalent to the VBA Save command in Excel. The workbook.Save method syntax is as follows: expression. Save where the object expression is the corresponding task you want to save. Let's take a look at an example to make this clearer. The following macro with Save_Workbook name saves current active workbook: This excel VBA save workbook tutorial comes with an Excel workbook containing the data and macros I use (including Save_Workbook macros). You can get instant free access to this workbook for example by subscribeing to the Power Spreadsheets newsletter. Note that the macro has just 1 statement that follows the general syntax of the workbook method.Save described above: ActiveWorkbook.Save in this case, ActiveWorkbook is a simplified reference to application.ActiveWorkbook property. This property will bring back a workbook object, based on the Workbook.Save method. The workbook returned by the ActiveWorkbook property is, more precisely, the workbook in the current active window. In short, the macro Save_Workbook will simply save the current active Excel workbook. As when working directly with Excel, the Save method is an important command/method that is relatively easy and straightened out. However, it does not allow you to determine much saved in connection with the relevant Excel way of working. The workbook is saved and that's pretty much it. When working directly in Excel, you will be using the Save As if you want to be able to determine more about the way the actual savings are made from a workbook. Work in a similar fashion within the basic visual for applications. To be more precise, when working with Visual Basic for Applications, you can use the SaveAs method for these purposes. So let's take a look: How to save Excel workbook using workbook. SAVEAS VBA reasoning method or parameters is a method of something that allows you to determine the action features that a specific method does. As described above, the Workbook.Save method has no parameters. As a result, you can't really determine much about how the corresponding workbook is saved. The Workbook.SaveAs method is different. Its 12 parameters allow you to determine more different aspects about the way in which excel workbook is saved. In other words, Workbook.SaveAs is more flexible and complex than Workbook.Save. The workbook. SaveAs is, almost speaking, the VBA equivalent of the Save As command in Excel. So, it allows you to save a workbook in a specific file. The full syntax of the Workbook.SaveAs method is as follows: expression. SaveAs(FileName, FileFormat, Password, WriteResPassword, ReadOnlyRecommended, CreateBackup, AccessMode,ConflictResolution, AddToMru, TextCodepage, TextVisualLayout, Local) expression is, just as in the case of the Workbook.Save method above, the relevant Workbook object. All parameters (that appear within the pronths) are optional from the SaveAs method. However, in order to understand what this method can do to help you, I will explain these parameters below. However, as usual, I use a practical macro example for purposes showing how Workbook.SaveAs works. So let's start by looking at the basic VBA code of macro example: how to save an Excel workbook with new name using workbook. SaveAs method the following piece of VBA code save current active workbook with new name provided by the user. Dim workbook_Name As Variant workbook_Name = Application.GetSaveAsFilename If workbook_Name <> False Then ActiveWorkbook.SaveAs Filename:=workbook_Name End If The following screenshot shows the VBA code behind the example macro (called Save_Workbook_NewName) which is included in the Excel workbook that accompanies this Excel VBA Save Workbook Tutorial. You can get instant free access to this workbook for example by subscribeing to the Power Spreadsheets newsletter. This macro can be divided into 3 sections: Let's take a quick look at each of these sections to find out how macro Save_Workbook_NewName works: Part #1: Dim workbook_Name As Variant This statement simply announces a variable called workbook_Name. The variable is the type of variant data. Although the variant variables are sometimes undesirable, in this particular case it is not necessarily the case. The species variable allows GetSaveAsFilename (which I introduce below) quite flexible. As it implies with its name, and revealed by the following sections of macro, the purpose of the workbook_Name new save Saved from Excel workbook. Part #2: workbook_Name = Application.GetSaveAsFilename This statement will change the value workbook_Name variable. Which value is actually assigned is determined by application.GetSaveAsFilename method, which I will explain in full below. At its most basic level, the GetSaveAsFilename method does the following 2 things: Step #1: Displays the Save As dialog box. You are probably quite familiar with this dialog box, as it is one of excel views when you run the Save As command. Step #2: When the user saves a file name through the dialog box as provided, GetSaveAsFilename gets that specific name. This is the name that the entire statement we are analyzing has workbook_Name. Note that the Application.GetSaveAsFilename method does not actually save a file. It simply becomes a name. To save the actual file using the name provided by the GetSaveAsFilename method, you usually rely on the Workbook.SaveAs method. This method is used Save_Workbook_NewName in the last #3: Part workbook_Name <> False Then ActiveWorkbook.SaveAs Filename:=workbook_Name End If this is an If... Then... Another statement. These types of statements are conditionally executed by a specific group of statements, depending on whether or not a condition is met. The statement begins with the word If. The entire block ends with an End If statement. In the case of Save_Workbook_NewName if... Then... Another statement goes as follows: Step #1: Test whether the workbook_Name >> is incorrect. Part one, uh... Then... Another statement conducts a logical test. This logical test seeks to confirm whether workbook_Name has a value that differs from (lt;>) of the false logical value. If the workbook_Name is not incorrect, the logical test (workbook_Name >>) will be evaluated to True. In such a case, the statements within if... Then... They are executed again. However, if the value workbook_Name the amount of Boolean False, the logical test will be evaluated to False. In this case, conditional statements are not executed. For the purposes of this logical test, the workbook_Name of the variable is the number allocated in the previous section. So the value depends on the input given by the user when displaying the Save As dialog box. Put more precisely: If the user cancels the Save As dialog box, the value workbook_Name false. If the user provides a file name through the Save As dialog box, the variable value workbook_Name (generally) assigned by the user. In other words: if the user offers a file name: the logical test performed by the first part if... Then... Another statement is true, and the probation statements that follow are carried out. If the user cancels the Save As dialog box (e.g. clicking the Cancel button): the test is logical false; Then... Another statement is not executed. Step #2: Execute the Statement Filename:=workbook_Name If The Tested Condition Is True. You already know that, almost speaking, the logical test workbook_Name >> incorrectly returns correctly if the user saves a file name through the dialog box as assigned. In this case, the following statement is executed: ActiveWorkbook.SaveAs Filename:=workbook_Name This is where the Workbook.SaveAs method comes into play. This statement does the following: Step #1: Uses the Application.ActiveWorkbook property to return the workbook in the current active window. Step #2: Saves the active workbook in a file named by the user through the Save As dialog displayed by the GetSaveAsFilename method. In this particular case, only 1R argument is used from the Workbook.SaveAs method: Filename. The file name argument, as implicitly specified by its name, allows you to specify the name of the saved workbook. I have a further explanation of the file name arguments, and other arguments of the SaveAs method, in the following sections. If the tested status is not correct, no other statements will be executed. In other words, the workbook will not be saved when it has been unused in the Save As dialog box. The Workbook.SaveAs Method: Parameters The following table introducs the 10 most important optional parameters of the Workbook.SaveAs method: PositionNameDescription 1FilenameName of saved workbook. 2FileFormatFile format for saved workbook. 3PasswordProtection password for saved workbook 4WriteResPasswordWrite - Reserve password for saved workbook. 5ReadOnlyRecommendedDetermines whether the workbook is saved is recommended as just reading. 6CreateBackupDetermines whether a backup file of the saved workbook has been created. 7AccessModeDetermines the access mode of the saved workbook. 8ConflictResolutionApplies only if the saved workbook is shared. Determines how conflicts they show when saving are resolved. 9AddToMruDetermines whether the saved workbook has been added to the list of recently used files. 12- Determines whether the workbook is saved against Excel language (usually local) or VBA (usually US-English). 2 Parameters of saveAs method (#10, TextCodepage and #11, TextVisualLayout) are not included in the table above, nor are explained below. According to Microsoft's official documentation (at the time of writing), both of these parameters are ignored. Let's take a closer look at each of the individual arguments of the workbook. SaveAs: #1: File name as implicit with its name, you are using the file name argument of the Workbook.SaveAs method to specify the saved workbook name. When working with Filename argument, you can either specify the full file path, or do not specify the file path. If you do not specify the file path, Excel will save the workbook to the current folder. For most users, it's not so easy to specify the file path. You (or user) need to specify the exact file paths, names and extensions. The approach is tedious and error-prone. This is the main reason why Application.GetSaveAsFilename used in Very useful: it allows the user to browse different folders and easily specify the full file path and the name of the Saved Excel workbook. The basic basic Save_Workbook_NewName macro uses filename argument R, as shown in the following screenshot: Argument #2: FileFormat you can use the FileFormat argument of the Workbook.SaveAs method to specify the saved file format. If you don't use fileFormat argument, Excel will specify the file format as follows: In the case of workbooks already existing, the workbook will be saved using the same file format last time. If the workbook is new, the workbook will be saved using the Excel version format you use. Even if this parameter (as all other arguments of the SaveAs method) is optional, you may want to develop a habit of using it. You specify a specific file format using the XlFileFormat measurement. Microsoft's developer network lists more than 50 different possible values. In practice, you are unlikely to need/use many different formats. Actually, some of the formats listed in Microsoft's developer network are not supported in recent versions of Excel. So, I provide a basic overview and parse the value of XlFileFormat that you may actually encounter. Even if this list is much shorter than that on Microsoft's developer network, you're still likely to use only a subset of my values below to explain. Below are 4 original file formats in Excel 2007-2013: 50:xlExcel12. 51: xlOpenXMLWorkbook. 52: xlOpenXMLWorkbookMacroEnabled. 56: xlExcel8. As a general rule, it is best to use FileFormat values (numbers) instead of names. The reason for this is that this avoids some consolidated problems whenever you run the corresponding macro in an older version of Excel that may not recognize the name. So let's take a look at some of the value that FileFormat's argument can be: ValueNameDescription Add-ons and Templates 17xlTemplate/xlTemplate8Template/Template 8. It is generally used in versions between Excel 97 and Excel 2003. 18xlAddIn / xlAddIn8Excel 1997 to 2003 Add-In. 53xlOpenXMLTemplateMacroEnabledMacro-Enabled Open XML template. 54xlOpenXMLTemplateOpen XML template. 55xlOpenXMLAddInOpen XML Add-In. Text files -4158xlCurrentPlatformText file format for the platform where the workbook is stored. 2xlSYLKSymbolic file format link. Only the active sheet is saved. 6xlCSVCSV (values separated from commas) text file format. 9xlDIFData Interchange Format file. Only saves the current active sheet. 19xlTextMac text file format. Ensures that basic formatting (such as tabs and line breaks) and characters are interpreted correctly. xlTestMac only saves the active sheet. 20xlTextWindowsWindows text file format. Ensures that basic formatting (such as tabs and line breaks) and characters are interpreted correctly. xlTestWindows only saves the active sheet. 21xlTextMSDOSMSDOS TEXT FILE FORMAT. Ensures basic formatting (such as and line breaks) and characters are interpreted correctly. xlTestMSDOS only saves the active sheet. 22xlCSVMacCSV file format for Mac platform. Ensures that basic formatting (such as tabs and line breaks) and characters are interpreted correctly. xlCSVMac only saves the active sheet. 23xlCSVWindowsCSV file format for Windows platform. Ensures that basic formatting (such as tabs and line breaks) and characters are interpreted correctly. xlCSVWindows only saves the active sheet. 24xlCSVMSDOSCSV File Format for MS-DOS platform. Ensures that basic formatting (such as tabs and line breaks) and characters are interpreted correctly. xlCSVMSDOS only saves the active sheet. 36xlTextPrinterFormatted text file. Only saves the current active worksheet. 42xlUnicodeTextUnicode text file format. Spreadsheets (Excel and others) -4143xlWorkbookNormalExcel file format. 39xlExcel5/xlExcel7Excel versions from 1993 (Excel 5.0) and 1995 (Excel 7.0). Versions 43xlExcel9795Excel from 1995 and 1997. However, as explained by author Richard Mansfield in VBA Mastering for Microsoft Office 2013, this file format is generally compatible with Excel 95 versions and later. 46xlXMLSpreadsheetXML spreadsheet file format. Generally used in Excel 2003. 50xlExcel12Excel 2007 edition. 51xlOpenXMLWorkbook / xlWorkbookDefaultOpen XML workbook / Workbook default file format. 52xlOpenXMLWorkbookMacroEnabledMacro-Enabled Open XML workbook. 56xlExcel8Excel version from 1997. 60xlOpenDocumentSpreadsheetOpen Document Spreadsheet file format. OpenDocument spreadsheet files can be opened using spreadsheet applications that use openDocument spreadsheet format. Examples of such programs include Google Tabs, Open Office Calc and Excel itself. Formatting may have been affected when saving or opening open document spreadsheet files. 61 (&H3D)xlOpenXMLStrictWorkbookISO Strict Open XML file format. Clipboard Files 44xlHtmlHTML / webpage file format. If you save excel workbook to CSV or text file format, below 2 things happen: Excel selects the code page to use by checking the local system settings on the computer where the workbook is stored. The code page used is one corresponding to the language for the local system in use. In Windows 10, you can find these settings by going to Settings > Time & Language > Region & Language. Excel saves the file in logical layout. This is relevant, specifically, when working with files containing two-directional text, where text may be in different directions (left to right and right to left). Whenever text is embedded in one direction within the text in the other direction, it stores the logical layout of the file in such a way that the reading command is correct for all languages that are used, regardless of their orientation. Then when such a file opens later, all text within the file (in general) will be displayed in the appropriate order. This direction is determined by the character value ranges of the code page in use. let's go I'm Save_Workbook_NewName. The following screenshot shows how the VBA code of this macro looks like when I add fileFormat argument and set its value to 52 (the macro enables the open XML workbook). Argument #3: Password argument of the Workbook.SaveAs method allows you to (as you might expect) to enter the password to protect the saved Excel workbook. R The password argument has three main features: a string. Sensitive to the file. The maximum length is 15 characters. The following screenshot shows the VBA code on the back Save_Workbook_NewName macro with password. In this case, the password is excel tutorial. If you save a workbook by using macros such as above, next time anyone (you or another user) tries to open the Excel workbook, Excel will display the password dialog. If the wrong password has been entered, Excel will not open the workbook. It displays a warning instead. The #4: WriteResPassword is the WriteResPassword parameter of the Workbook.SaveAs method, in some respects, similar to the password argument that I explain above. However, Password and WriteResPassword differ in an essential feature: they protect different things. As described above, the password protects the workbook. If you (or corresponding user) failed to provide the correct password, Excel will not open the workbook. WriteResPassword protects the workbook writing booking feature. To see what this is, and how it works in practice, I add writeResPassword's argument Save_Workbook_NewName macro. The password for these purposes is Excel Course. The dialog box that Excel displays to apply WriteResPassword is slightly different from the box it uses when asking for a password. Notice how it informs that the saved user has saved it and provides 2 options: You can enter the password and excel will access the allowance you write. Otherwise, you can open the workbook as read-only. If I choose to open the workbook as reading only, Excel does exactly so. In that case, it warns in several places that the workbook is read-only and changes are not saved. If you enter the wrong WriteResPassword, Excel will react in the same way when you enter the wrong password (as shown above). In other words, it doesn't open the workbook and displays the following message: #5 Argument: ReadOnlyRecommended ReadOnlyRecommended argument saves you with a less rigorous way (when compared with the above WriteResPassword) to protect your Excel workbook. More precisely, if you recommend a specific workbook for reading only, Excel displays messages making such recommendations whenever the file is opened. Setting the workbook for reading is only recommended not actually protecting or saving the workbook in the same way the password or WriteResPassword do. Each user can open a normally recommended Excel workbook (not as read-only) by, for example: clicking No in the dialog box. Top. organization argument of the Workbooks.Open argument to True when opening the workbook using VBA. To determine which Excel workbook is recommended reading only, you simply set the ReadOnlyRecommended argument to be true. Argument #6: CreateBackup createBackup argument of the Workbook.SaveAs method allows you to determine if a backup of the workbook is being saved. If you want to create a backup of the saved Excel workbook, set the CreateBackup argument correctly. Argument #7: AccessMode Argument AccessMode allows you to specify access mode for saved workbooks. This argument R can take the following 3 values: 1: stands for xlNoChange. In this case, the default access mode is used. 2: XlShared represents. In this case, access mode is the subscription list. 3: Value for xlExclusive. In this scenario, access mode is the exclusive mode. The following image shows the VBA code of Save_Workbook_NewName macro with accessMode parameter set to xlNoChange: argument #8: ConflictResolution ConflictResolution applies when you are working with the shared workbook. More precisely, this argument allows you to determine how conflicts (while saving Excel workbook) are solved. You can set the ConflictResolution parameter to any of the following 3 values: 1: abbreviation xlUserResolution. In this case, Excel displays a dialog box which will ask the user to solve the conflict. This is the default setting if you remove conflictResolution arguments. 2: XlLocalSessionChanges. If you choose this value, changes made by the local user will always be accepted. 3: Value for xlOtherSessionChanges. This is the opposite of the above: changes made by the local user are always rejected. The following image shows the macro code Save_Workbook_NewName conflictResolution parameter set to the default xlUserResolution. Argument #9: AddToMru MRU stands for Most Recently Used. This makes reference to the Excel list of recently used files that, in general, you'll find in the behind-the-scenes view. AddToMru's argument of the Workbook.Save method will allow you to determine if the saved workbook has been added to this recently used list. If AddToMru is set to True, excel workbook will be added to the list. The default value of AddToMru, however, is False. In the following screenshot, you can see the VBA code behind the sample Save_Workbook_NewName macro with AddToMru set to the correct: as mentioned above, I will not cover the arguments in the textCodePage and TextVisualLayout details (#10 and #11 arguments). Local #12 is the latest argument of the local Workbook.SaveAs method. As implicitly saved by its name, local refers to the language and localization aspects of the workbook. More precisely, the local parameter allows you to determine if the saved workbook is saved against the language: Excel, generally determined from the Control Panel setting; The basic exception to this rule is the VBA language being used in English When the VBA project that runs the Workbook.SaveAs method is an international XL5/95 VBA project. My guess is that you are unlikely to often work with such projects. To determine how Excel will go in relation to this topic, you can set the local argument to True or False. Correct: Saves workbook against Excel language. False: Saves Excel workbook against VBA language. On the following image, you can see the Save_Workbook_NewName sample with the Local set to True parameter: How to Save A Copy of An Excel Workbook Using The Workbook.SaveCopyAs VBA Method The Save and SaveAs methods explained above are the basic methods you'll need to save Excel workbooks using VBA. However, both of these methods are saved and changed now open excel workbook. You may have encountered some situations where this is the result you want. In other words, you'll probably be in the condition that you want a macro simply: Save a copy of current Excel workbook, but... Don't actually change the current file in your computer's memory. These types of positions are great for using the Workbook.SaveCopyAs VBA method. This method does exactly that. Takes the workbook and: Saves a copy to a file. It doesn't correct in memory. The syntax of the SaveCopyAs method, once again, is fairly simple: expression. SaveCopyAs (file name) just as with other methods explored in this Excel tutorial, the expression represents the working object. The file name, the only parameter of the SaveCopyAs method is the full file path, name and copy extension that you save. Since you are likely to use this method in active workbook most of the time, you will probably end up using the following syntax often: ActiveWorkbook.SaveCopyAs (file name) another commonly used alternative is to use thisWorkbook property instead of ActiveWorkbook. The main difference between ThisWorkbook and ActiveWorkbook is: ActiveWorkbook refers to the current active workbook. ThisWorkbook refers to the workbook where the macro is actually stored. Let's take a look at an example of the macro that uses the Workbook.SaveCopyAs method to save a copy of current active workbook: the following screenshot shows the macro called Save_Copy_Workbook. This macro has a single (quite long) statement. This goes as follows: ActiveWorkbook.SaveCopyAs Filename:=ActiveWorkbook.Path & \Copy & Format(Now, yy-mm-dd) & & ActiveWorkbook.Name Notice that the structure I use in the

Save_Copy_Workbook macro follows the basic syntax of the Workbook.SaveCopyAs method explained above. However, let's split the statement into 2 sections to better understand what's going on, and what this particular method can do for you: Part #1: ActiveWorkbook.SaveCopyAs This refers to the SaveCopyAs method. Follows the basic syntax described above. ActiveWorkbook mentions application.Workbook property. This property returns a workbook object indicating the current active workbook. The active workbook is manipulated by the SaveCopyAs method. In other words, the statement simply tells Excel to go ahead as follows: step #1: Consider the current active workbook. Step #2: Save a copy of current active workbook, without actually changing it in memory. Part #2: Filename:=ActiveWorkbook.Path & \Copy & Format(Now, yy-mm-dd) & & ActiveWorkbook.Name This part of the statement specifies the only argument of the Workbook.SaveCopyAs method: The Filename. This particular file name is a little long to copy but, essentially, is made by concatenating 5 cases. You use the Ampersand operator (&) to categorize different items. Item #1: ActiveWorkbook.Path This makes reference to the Workbook.Path property. The property returns the full route to the relevant record. In the example above, ActiveWorkbook.Path is used to get the path to the current active workbook. Let's say, for example, the current active workbook (called Book1) is saved to drive D. In this track mode, simply D:. This sample path (D:) It's not too long or complicated however, in practice, you're more likely to work with longer and more complex routes that you just work with drive D. The #2 #4: \Copy and these are, simply, text strings. The first string specifies which is the first word in the Copy file name. The second string adds a space(). Item #3: Format(Now, yy-mm-dd) This particular statement uses 2 VBA built-in functions, as follows: Now returns today's date and the current time. Alternatively, you can use the Date function that returns the current date. The format takes the date restored by Now and formats it according to the yy-mm-dd date format. In other words, this part of Argyll is responsible for the historical return in which the copy is stored in yy-mm-dd format. For example, if the date on which you save a copy of the workbook is November 30, 2015, this item goes back 15-11-30. Item #5: ActiveWorkbook.Name uses the Workbook.Name property to get the name of the workbook. For example, if the workbook name is the best Excel tutorial, Workbook.Name returns to it. In order to clarify everything about the workbook.SaveCopyAs method, let's take a look at an example: how to save a copy of Excel workbook using the workbook. SaveCopyAs VBA Method: One example to assume that the current active workbook is called the best Excel tutorial and is stored in drive D (D:Save_Copy_Workbook Save_Copy_Workbook). Let's back to the Filename argument of the SaveCopyAs method used within the Save_Copy_Workbook macro: Filename:=ActiveWorkbook.Path & \Copy & Format(Now, yy-mm-dd) & & Notice how, each of the 5 items described above is your expression in action when the macro runs: Item #1: Copy is stored in the same folder as the original workbook, as given by workbook.Path property. Items #2 #4: The first word in the actual workbook name is copied, as determined by the string \Copy. Also, there is a space between the date (15-11-19) and the original workbook name (best Excel tutorial) as specified by. Item #3: The date on which the workbook is stored (November 19, 2015 in the example above) is added in the form of yy-mm-dd (15-11-19). Item #5: The original workbook name (best Excel tutorial) has been added at the end of the copy name. The following screenshot shows this: How to name a workbook using Application.GetSaveAsFilename Method I introduced application.GetSaveAsFilename method above. This method is used by one of the sample macros (Save_Workbook_NewName) to open the Save As dialog box, allowing users to browse easily and insert the path, name and suffix of the saved Excel workbook file. The following screenshot shows the VBA code of Save_Workbook_NewName macro. Notice the application.GetSaveAsFilename method. Application.GetSaveAsFilename method does not actually save a file. However, GetSaveAsFilename is a useful method to use whenever you have a macro that needs to get a file name from the user in order to, among others, save a workbook. GetSaveAsFilename is useful when the method requires receiving/knowing the file name to save. This allows the user to specify the file path and file name. As I explain below, you can use the Application.GetSaveAsFilename method for exactly these purposes. The GetSaveAsFilename method has a few parameters that allow you to customize some of its features. Let's take a closer look at your method and its arguments, starting with: Application.GetSaveAsFilename Method: Target Application.GetSaveAsFilename Method does 2 things: view save as dialog box. Gets the file name that the user entered in the Save As dialog box. GetSaveAsFilename does not save a workbook by itself. That's why, for example, the Save_Workbook_NewName above macro includes using the Workbook.SaveAs method to save the actual Excel workbook. Application.GetSaveAsFilename Method: Syntax The full syntax of the Application.GetSaveAsFilename method is as follows: expression. GetSaveAsFilename(InitialFilename, FileFilter, FilterIndex, Title, ButtonText) expression is used to represent the Application object. So, you, most likely, usually use the following basic syntax for this method: Application.GetSaveAsFilename This syntax used in the version of the Save_Workbook_NewName method shown above. All 5 Argom GetSaveAsFilename methods are optional. Let's take a look at them: Application.GetSaveAsFilename Method: The following table argument provides a basic Of the 5 parameters of the Application.GetSaveAsFilename method. I will explain each of them more thoroughly below. PositionNameDescription 1InitialFilenameSpecifies a suggested/default file name. 2FileFilterDetermines file filtering criteria. 3FilterIndexDetermines default file filter. 4TitleDetermines the title of the (usually called) Save As dialog box. 5ButtonTextApplies only on the Mac platform. Specify text (normally called) save as button. There are quite a few similarities between the GetSaveAsFilename method and the GetOpenFilename method (which I describe here). In terms of their arguments, the main differences are: GetSaveAsFilename has a PrimaryFilename argument. GetOpenFilename does not. GetOpenFilename has a MultiSelect argument. GetSaveAsFilename does not. Both of these differences make sense. For example, MultiSelect allows you to determine whether a user can select multiple file names at the same time. This makes sense in the context of opening files. But not in the field of storing files with getSaveAsFilename method. Let's take a look at each of the parameters introduced above: #1: InitialFilename is the initialFilename name of the program. GetSaveAsFilename method allows you to set a suggested file name. This suggested file name is the file name that appears as in the Save File name box by default. The Save As displayed above dialog box is the result of running the following version of Save_Workbook_NewName macro. Notice that the InitialFilename argument has been added and the suggested name is the best Excel tutorial, as shown in the screenshot above. Argument #2: FileFilter's fileFilter argument of application.GetSaveAsFilename method allows you to set file filter criteria in the Save As dialog box. These file filtering criteria will determine what appears in the Save As type drop down list dialog box of Save As dialog box. If you delete fileFilter's argument, the default (as shown in the below screenshot) is All Files. This is not ideal because it may lead to excel workbook being saved from indestrusible file type if the user does not enter file extensions when saving the file. However, my guess is that you will be in a situation where it is easier or even necessary to specify file filtering criteria. To determine which file filters appear in the Save As dialog box, you need to follow the following 4 instructions. Don't worry if the instructions don't seem that clear at first. I will show you a practical example of VBA code after making the introduction and basic description. Guide #1: Each filter is composed of a pair of strings. Each filter you specify is made from 2 comma separated strings when using fileFilter argument R. This seems, almost, as follows: String1, String2 String1 and String2 have different structures and goals. More precisely: String1: a descriptive string. This string specifies what is actually in Save as type drop down box from the Save As dialog box. String2: The filter specification is the file type of MS-DOS wildcard. In other words, this string determines how files are actually filtered depending on their file format. You don't have to follow many instructions on how string1 is specified. However, you need to look for a more specific syntax when specifying the second string (String2). Let's take a look at it: #2: Syntax to specify file type filters. The second string that you use to specify your file filter is composed of 3 elements that, in general speaking, are as follows: element #1: asterisk (*), used as wildcard. Element #2: Dot (.). The #3: a sign of the file extensions used to filter files. This particular element is usually composed of (where appropriate) a star (*), used as a wildcard, and/or (if appropriate), some text. The most basic filter is all files, which in practice means there is no filter. To specify a filter of file type from includes all files using the syntax above, you'd like to type asterisk dot asterisk (*.*). Other examples of filter specifications of the file type followed by this syntax include *.txt for text files. *.xla for add-ons. *.xlsx for Excel workbook. *.xlsm for Macro-Enable Excel workbooks. *.xls Excel 97 to Excel 2003 workbooks. *.csv CSV files. Knowing these first 2 instructions is enough for you to start using fileFilter arguments. However, they only explain how to specify a single filter according to a single file type. However, when working with FileFilter, you can actually specify: multiple different filters; 2. The next instruction shows how to get each of these done: Guide to #3: Syntax to specify multiple filters. You can create more than one filter with FileFilter argument. To do this, use commas (,) to separate filters. In other words, by comma (,) separate each pair of strings that form one filter from the pair of other strings. This looks, roughly, as follows: String1Filter1,String2Filter1,String1Filter2,String2Filter2 Guideline #4: Syntax To Specify Several File Types In A Single Filter. If you need to filter according to multiple different data types, you can use a few filters using the syntax described above. Alternatively, you can specify multiple data types for a specific single filter. To do this, use the MS-DOS wildcard phrases you use with toxic clones (;). This looks almost as follows: String1,String2.1; String2.2 Those 4 basic instructions you need to keep in mind to start using fileFilter arguments. Let's go back Save_Workbook_NewName macro and create some file filters: show the following screenshot (again) vba code Save_Workbook_NewName. Note that fileFilter's argument is inserted and its syntax follows all the instructions I explained above. To make this. Let's break the value of the argument into its different parts and highlight how it matches all the instructions described above. The full argom is as follows: Excel Workbook,*.xlsx,Excel Macro-Enabled Workbook,*xlsm,Excel Templates,*.xltx;*.xltm Attention to the following: There are 3 filters. Each filter is separated from the other by comma (),. Each filter is composed of 2 parts: a descriptive string and the specifications of the corresponding MS-DOS wildcard file type filter. These two episodes are separated by commas (,). MS-DOS wildcard file type Filter specifications Looking for syntax described above: (i) asterisk (*); (ii) dot (.); and (iii) file extension specifications, no wildcard asterisks in this case. The last filter uses 2 different file types. These types of files are by a semi-clone (;). The following screenshot shows how all of the above appears in #3. This is the first filter that was identified with fileFilter argument R. However, you can change the default file filtering criteria by using filterIndex argument R. You do this by specifying the criteria index number you want to set as default. in the example above) take. If you set the FilterIndex value to a number higher than the value of existing filters (4 or Save_Workbook_NewName in the case of Save_Workbook_NewName macro), the first filter will be used. In other words, the practical result of specifying an index number that is too high is the same result as removing the FilterIndex parameter. The following image shows the macro code Save_Workbook_NewName filterIndex parameter set to 2. In the case of this macro, a FilterIndex 2 value means that Excel Macro-Enabled Workbook is the new default filter. Argument #4: The title of the argument of the Application.GetSaveAsFilename method allows you to change the title (usually called) save as dialog box. If you delete the argument, the default title (Save As) is retained. The following screenshot shows how to use this argum to change the title of Save As dialog box when running Save_Workbook_NewName macro. In this case, the title argum is set to VBA save Excel workbook. When this macro runs, (formerly called) save as dialog pops out as follows. Note that the title has actually been changed to VBA Save Excel workbook. Argument #5: ButtonText buttonText parameter is only applicable on the Mac platform. If you use this argument in Windows, it's simply ignored. For them. Where applicable, the Argument ButtonText allows you to set the text that appears in the (commonly known) Save button. Conclusion Knowing how to save Excel workbook with VBA is essential. If you have read this Excel tutorial, now know the basics of how to save workbooks with VBA. Actually, you've seen 3 different ways to achieve this: using the Workbook.Save method. Using the Workbook.SaveAs method. Using the Workbook.SaveCopyAs method. Each of these items is explained with the help of a real example of VBA code. In addition, in the last part of this blog, I explained the Application.GetSaveAsFilename method. Although this method doesn't actually save a file by itself, it allows you to display the Save As dialog so that your macro users can easily specify the path and file name of the workbook they store. Save.