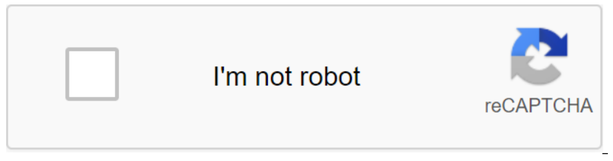
Excel vba activeworkbook save as pdf



Continue

Fragments on this page and in the list below are included in the VBA code library. Please note that the library contains much more than shown here. Also, check out the VBA Demo code below the custom VBA Search Code VBA Demo Click on the image below to start the VBA demo screencast code to get a quick impression of what it will do for you (49 seconds, 600kb). The Worbook SaveAs method is used to save changes in the work book in another file, such as Dim wb As Workbook: Set wb and Dim strFilename As String: Filstrename and wb. SaveAs Filename: 'strFilename, FileFormat: 'xlOpenXMLWorkbook On this page we show the various options available while saving the work book using the arguments SaveAs Filename FileFormat, Password, ReadOnlyRemendedcom, CreateBackup, AddToMru and Local. Filename file name to be saved. You can turn on the full path; If you don't Excel saves the file in the current folder. Please note that the extension used with the file name should correspond to the file format - see the table below FileFormat File format for use when saving the file. For an existing file, the default format is the last file format; The default version format for the new file is Excel. The most commonly used values are xlWorkbookNormal (-4143) excel xls 1997 - 2003 xlOpenXMLWorkbook (51) xlsx without macro xlOpenXMLWorkworkbookMacroEnabled (52) xlsm with or without macro xlEx cel12 (50) xlsb Excel Binary Workbook with or without macro xlExcel8 (56) xls Excel 2007 or later xlCSV (6) csv Local: 'True For the full list of valid options see XlFileFormatenumeration. Password A, a case-sensitive line (no more than 15 characters), which indicates a security password that must be passed to the file. WriteResPassword Line, which indicates the record-booking password for this file. If the file is stored with a password and the password is not supplied when the file is open, the file is open as soon as read. ReadOnlyRecommended True to display a message when a message is open file with the recommendation to open the file as soon as reading (the default is false). CreateBackup True to create a backup file (the default is false). AccessMode - Share or Exclusive Access Mode for a work book: shared or exclusive. xlExclusive (3) Workbook can only be edited by one person at a time, xlShared (2) Workbook can be edited by multiple individuals at the same time xlNoChange (1) The current value for the work book will be supported by ConflictResolution determines the way conflicts need to be resolved whenever the general book is updated. xlLocalSessionChanges (2) Changes to the local user are always accepted. xlOtherSessionChanges (3) Changes local user always reject. xlUserResolution The dialog asks the user to resolve the conflict. If this argument is omitted, you'll see a conflict resolution dialogue window. AddToMru True to add this work book to the list of most recently used the default is false. Local True saves files in Microsoft Excel (including control panel settings). False (default) saves files in visual basic for applications (VBA) (which is usually U.S. English if the VBA project is whereworkbooks. Open works from the old internationalized PROJECT XL5/95 VBA). CODE VBA - AGORA Software BV Copyright 1997-20166 VBA save as Workbook Excel Macro code helps keep the file in a particular Folder, its a common challenge in the automation process. Once you're done with an actual calculation or task, at the end of the procedure we usually call the procedure of exporting or saving the output file to a specific folder or shared disk. Or otherwise, you may not have permission to keep the file in place, so you can use the SaveAs option to store a revised or updated file. VBA Save as Workbook - Solution (s): You can use the SaveAs method to keep the file in a specific location. You can save the file with the same name and location. Or you can use different file names and location to save the file. You can also tune in to the object and save the file. Another method you use is Save Dialog Box. In this way, the user can select a specific folder to save the Excel file. The following example shows you how to keep the Excel Workbook book in a specific folder using SaveAs: Sub ExampleToSaveWorkbook () Workbooks.Add 'Saving Workbook ActiveWorkbook.SaveAs C: WorkbookName.xls' or 'ActiveWorkbook.SaveAs Filename: C:WorkbookName1.xls End Sub Set If you're dealing with more than one work book, you'll need this method to access the workbook. Sub ExampleToSaveWorkbookSet () Dim wkb How Workbook 'Adding New Workbook Set wkb and Workbooks.Add' Saving Workbook wkb. SaveAs C:WorkbookName.xls 'OR 'wkb. SaveAs Filename: C:WorkbookName1.xls End Sub You can save the working book for a specific Folder by showing Save the Dialogue Box to the user. For the user to choose the right place to save the file. Sub sbSaveExcelDialog () Dim IntialName As String Dim sFileSaveName As Variant IntialName - Sample Output sFileSaveName - Application.GetSaveAsFilename (InitialFileName: 'InitialName, fileFilter: Excel Files (.xlsm), y.xlsm) If sFileSaveName is then a false then ActiveWorkbook.SaveAs sFileSaveName End If the end of Sub you can keep the work book in the same macro book catalog using ThisbookWork.Path Property. Sub ExampleToSaveWithSamePathDifferentName () Dim sFilename As String sFilename - WorkbookName.xls You can give it to save Workbooks.Add 'Saving Workbook ActiveWorkbook.SaveAs ThisWorkbook.Path Sub Example ToSaveWithSameNameandPath () 'Saving the work book End Sub You Can file example and explore it. ANALYSISTABS - Save workbook Rewrite an existing work book using VBA While saving an existing work book or a new Excel file with an existing name, Excel will prompt a warning message. It interrupts the procedure and asks the user to click yes or no to rewrite the file. You can avoid this by temporarily disabling the alerts and save the work book with the same name by installing application.DisplayAlerts-False. Once you've completed the task, you should turn on the app alert by installing the TRUE property. The following example will show you how to rewrite a file by disabling your app alerts. Code: sub-procedure more write Excel file Sub ExampleToOverWriteExistingWorkbook () 'Declaration: Announcement of the Dim wkb Variable As Workbook' Adding a new workbook using Workbook.Add method and installation wkb Object Set wkb. SaveAs C:WorkbookName.xls 'change existng file name 'OR' wkb. SaveAs Filename: C: WorkbookName1.xls 'Eanbling app Aletrts after saving Application.DisplayAlerts - True End Sub Instructions: Open Excel workbook Press Alt-F11, to open the VBA editor Insert module to insert menu copy above the code and paste into the code window Save file as the macro included working book Press F5 to perform it Nu su nenhum especialista e vBA mas on ActiveWorkbook.Save and chamado dentro da auto_close. Entao, wai dependant de quando essa sub et chamada no cilco de vida do excel. Existem Formas de fechar o excel por linha de comando sem passar por essa fun'o. Por exemplo com o comando: ActiveWorkbook.RunAutoMacros xlRunAutoMacro.xlAutoClose Dei uma procurada na documenta'o e n'o encontrei como o excel fecha o software em caso de travamento. Mas, acredito que o padr'o do sistema operacional seja apenas matar a opera'o em caso de travamento. O que n'o passaria pelo trigger de fechar aplica'o que gente usa. O que faria com que ele n'o chamasse o m'todo e, por consequencia n'o salvasse a planilha. Espero ter ajudado. Abrano! If you've worked with Excel before, you're probably familiar with the 2 main teams for saving workbooks: It may not surprise you to know that when working with VBA, you can perform these same activities. In fact, knowing how to keep Excel workbooks using VBA is essential. Working with Visual Basic for apps, you'll notice that keeping work books is one of the most important things your macros can do. Due to the importance of knowing how to save workbooks with VBA, this Excel tutorial focuses on this particular topic: How to Save the Excel Book using VBA. In addition to presenting some examples of VBA code that can be used toworkbooks, I explain the basics surrounding the 4 VBA techniques that you probably encounter and use constantly while saving workbooks. The following content table shows specific topics that I will explain in this Excel tutorial: This Excel tutorial does not cover the topic of saving the Excel book as a PDF using VBA. I explain how to export the Excel file to the PDF using macros, and give a few examples of the code here. Let's start looking at the basic ways to save the Excel book with VBA. How to Save Excel Workbook Using Workbook.Save VBA Method The most basic method for saving Excel workbooks using VBA is the Workbook.Save method. Workbook.Save retains the relevant work book. In other words, the Workbook.Save method is, roughly speaking, the equivalent of the Save's Excel team VBA. The syntax of the Workbook.Save method is: expression. Save Where the Expression is an appropriate workbook object you want to keep. Let's look at the example to make it clearer. The next macro, named Save_Workbook, retains the current active work book: This Excel VBA Save Workbook is accompanied by an Excel work book containing the data and macros I use (including the macro Save_Workbook). You can get immediate free access to this exemplary work book by subscribing to the Power Spreadsheets newsletter. Note that the macro has only one statement, which follows the general syntax of the Workbook.Save method above: ActiveWorkbook.Save In this case, ActiveWorkbook is a simplified reference to the Application.ActiveWorkbook property. This property returns the Workbook object as required by Workbook.Save. The work book returned by activeWorkbook is, more precisely, a working book in the current active window. Thus, the example of Save_Workbook above simply retains the current active Excel work book. Just like working directly with Excel, the Save method is an important command/method that is relatively simple and easy to execute. However, this does not allow you to determine much in connection with how the corresponding Excel work book is saved. The work book is saved, and that's pretty much it. When you work directly in Excel, you use the Save As command if you want to be able to determine more about how the actual work book savings occur. Things work in a similar way in Visual Basic for apps. More precisely, when working with Visual Basic for applications, you can use the SaveAs method for this purpose. So let's take a look at: How to Save Excel Workbook Using Workbook.SaveAs VBA Method Arguments or Method Options, allowing you to determine the action characteristics that a particular method performs. As explained above, Workbook.Save does not No parameters. As a result, you can't really define much about The relevant work book has been preserved. The Workbook.SaveAs method is different. Its 12 parameters further define several aspects of how the Excel workbook is saved. In other words, Workbook.SaveAs is more flexible and complex than Workbook.Save. Workbook.SaveAs is, roughly speaking, the equivalent of the VBA of Save As's Excel team. Thus, it allows you to keep the work book in a specific file. The full syntax of the Workbook.SaveAs method is this: expression. SaveAs (FileName, FileFormat, Password, WriteResPassword, ReadOnlyRecommended, CreateBackup, AccessMode, ConflictResolution, AddToMru, TextCodepage, TextVisualLayout, Local) expression is, as in the case of Workbook.Save method above, the appropriate Workbook object. All options (which appear in brackets) of the SaveAs method are optional. However, in order to understand what this method can do to help you with, I'll explain these options below. However, as usual, I use a hands-on macro example to illustrate how Workbook.SaveAs works. So let's start by taking a look at the basic VBA macro-example code: How to save excel Workbook with a new name using Workbook.SaveAs Method The next part of the VBA code retains the current active work book with a new username provided by the user. Dim workbook_Name As a variant workbook_Name - Application.GetSaveAsFilename If workbook_Name of false then ActiveWorkbook.SaveAs Filename: Workbook_Name End If the next screenshot shows the VBA code behind the example of the macro (called Save_Workbook_NewName), which is included in the Excel workbook book that accompanies this Excel VBA Save tutorial. You can get immediate free access to this exemplary work book by subscribing to the Power Spreadsheets newsletter. This macro can be divided into the following 3 parts: Let's take a quick look at each of these parts to understand how the macro Save_Workbook_NewName: Part of the #1: Dim workbook_Name As Variant This statement simply announces a variable called workbook_Name. The variable has a type of Variant data. Although Variant variables are sometimes undesirable, this is not necessarily the case in this particular case. The variable allows GetSaveAsFilename (which I'll enter below) to be quite flexible. As its name suggests, and obviously from the following parts of the macro, the purpose of the variable workbook_Name is to store the new name of the saved Excel book. Part of the #2: workbook_Name and Application.GetSaveAsFilename This statement assigns a value workbook_Name variables. What value is actually assigned to Application.GetSaveAsFilename, which I'll explain in detail below. At the most basic level, the GetSaveAsFilename method does the following two things: Step #1: Displays save like a dialog box. You're probably good at dialog, as it's the one that displays Excel when doing the Save As command. Step #2: After a user has provided the file name through the Save As dialog box, GetSaveAsFilename receives this specific name. This is the name that the entire statement we analyze assigns a variable workbook_Name. Note that Application.GetSaveAsFilename doesn't actually save the file. He just gets a name. To save a file using a name provided by GetSaveAsFilename, you usually rely on Workbook.SaveAs. This method is used in the last part of the macro Save_Workbook_NewName: Part of the #3: If workbook_Name of False, then ActiveWorkbook.SaveAs Filename: workbook_Name End If it if ... Then... Another statement. These types of statements are conditionally performed by a certain group of operators, depending on whether the condition is fulfilled or not. The statement begins with the word If. The entire block ends with an End If statement. In the case of macro Save_Workbook_NewName, if... Then... Another statement goes on as follows: Step #1: Whether the workbook_Name is false. The first part of If... Then... Else's statement is a logical test. This logical test seeks to confirm whether workbook_Name is a value that differs from false's logical value. If the value of workbook_Name is not false, the logical test (workbook_Name of false) is rated as True. In this case, the statements in if... Then... Still being executed. However, if the value of the workbook_Name equals the value of Boolean False, the logical test is rated as false. In this case, conditional operators are not executed. For the purposes of this logical test, the value of the variable workbook_Name assigned in the previous part. Thus, the value depends on the input entered by the user when the Save As dialog window is displayed. More precisely: If a user cancels the Save As dialog, the value of workbook_Name is false. If a user provides a file name through the Save As dialog, the value workbook_Name the variable (usually) assigned by the user. In other words: If a user provides a file name: A logical test conducted by the first part of If... Then... Another statement is true; and conditional applications that follow are executed. If a user cancels the Save As dialog (for example by clicking on the Cancellation button): the logical test is false; and conditional statements within If... Then... No statement has yet been made. Step 2: Follow ActiveWorkbook.SaveAs Filename: workbook_Name if the test state is correct. You already know that, roughly speaking, the logical test workbook_Name False returns True if the user has assigned the file name through the Save As dialog window. In this case, the following statement is made: ActiveWorkbook.SaveAs Workbook.SaveAs comes into play here. This statement does the following: Step #1: Uses Application.ActiveWorkbook property to return the work book to the current active window. Step #2: Keeps an active work book in the file, whose name is given by the user through the Save As dialogue, displayed by GetSaveAsFilename. In this particular case, only one Workbook.SaveAs argument: Filename is used. Filename's name allows you to specify the name of the saved work book. I explain more about filename's argument, and other SaveAs method arguments, in the sections below. If the testing state is not true, further statements are not executed. In other words, the work book is not saved when used is cancelled save as a dialog window. The Workbook.SaveAs: Options Next table introduces the 10 most important additional parameters of the Workbook.SaveAs: PositionNameScription 1FilenameName saved workbook. 2FileFormatFile format for saved work book. 3Password Protector password for saved work book 4WriteResPasswordWrite-password booking for saved work book. 5ReadOnlyRecommendedDetermines Whether the working book is saved as read is only recommended. 6CreateBackupDetermines create whether a backup file of a saved work book is created. 7AccessModeDetermines Saved Work Book Access Mode. 8ConflictResolutionA applies only if the saved work book is shared. Determines how conflicts that occur when saved are resolved. 9AddToMruDetermines adds a saved work book to the list of recently used files. 12LocalDetermines, whether the work book is saved from the language Excel (usually local) or VBA (usually U.S.-English). The 2 SaveAs (#10, TextCodepage and #11, TextVisualLayout) are not included in the table above and are not explained below. According to Microsoft's official documentation (at the time of writing), both of these options are ignored. Let's take a closer look at each of the individual arguments of Workbook.SaveAs: Argument #1: Filename As its name suggests, you use the Filename argument of the Workbook.SaveAs method to indicate the name of the saved work book. When you're working with the Filename argument, you can also specify the full file path; or don't specify the file's path. If you haven't specified the file path, Excel saves the work book in the current folder. For most users, showing the file path is not very convenient. You (or the user) need to specify the exact paths of the file, names, and extensions. The approach is tedious and error-prone. This is the main reason why Application.GetSaveAsFilename used in Save_Workbook_NewName so useful: it allows the user to view different folders and easily specify the full file path and name of the saved book The original base version of the Save_Workbook_NewName uses Filename argument, as shown in the screenshot below: Argument #2: FileFormat You can use the FileFormat argument of the Workbook.SaveAs method to specify the file format of the saved file. If you don't use the FileFormat argument, Excel defines the file format as follows: In the case of existing workbooks, the work book is saved using the same file format as last time. If the work book is new, the work book is saved using the Excel version format you use. Although this option (like all other SaveAs arguments) is optional, you may develop a habit of using it. You specify a specific file format using the XlFileFormat listing. The Microsoft Developer Network lists more than 50 different possible values. In practice, you hardly need/use so many different formats. In fact, some of the formats listed on Microsoft's developer network are not supported in later versions of Excel. That's why I provide a basic overview and breakdown of the XlFileFormat values you may encounter. Although this list is much shorter than microsoft Developer Network, you'll probably only use a subset of values that I'll explain below. Below are 4 main file formats in Excel 2007-2013: 50: xlExcel12. 51: xlOpenXMLWorkbook. 52: xlOpenXMLWorkbookMacroEnabled. 56: xlExcel8. It is generally best to use FileFormat (numbers) instead of names. The reason for this is that it avoids some compilation problems whenever you perform the corresponding macro in the old version of Excel, which may not recognize the name. So let's take a look at some of the values that FileFormat argument may take: ValueNameDescription Add-Ins And Templates 17xlTemplate / xlTemplate8Template / Template 8. Usually used in versions between Excel 97 and Excel 2003. 18xlAddIn / xlAddIn8Excel from 1997 to 2003. 53xlOpenXmlTemplplateMacroEnabledMacro-Enabled Open XML template. Pattern 54xlOpenXmlTemplplateOpen XML. 55xlOpenXMLAddInOpen XML Add-In. Text files -4158xlCurrentPlatformText File format for the platform in which the work book is stored. 2xxlSYLKSymbolic Link file. Only the active sheet is preserved. Text file format 6xlCSVCSV (comma divided values). 9xlDIFDATA exchange file. Only the current active sheet is retained. Text file format 19xlTextMacMacMac. Ensures that the basic formatting (such as tabs and strings breaks) and symbols is correct. xlTestMac only retains an active sheet. 20xlTextWindowsWindows text file format. Ensures that the basic formatting (such as tabs and strings breaks) and symbols is correct. xlTestWindows only retains an active sheet. Text file format Ensures that the basic formatting (such as tabs and strings breaks) and symbols is correct. xlTestMSDOS only retains an active sheet. Sheet. file format for the Mac platform. Ensures that the basic formatting (such as tabs and strings breaks) and symbols is correct. xlCSVMac only retains an active sheet. File format 23xlCSVWindowsCSV for the Windows platform. Ensures that the basic formatting (such as tabs and strings breaks) and symbols is correct. xlCSVWindows saves only an active sheet. File format 24xlCSVMSDOSCSV for the MS-DOS platform. Ensures that the basic formatting (such as tabs and strings breaks) and symbols is correct. xlCSVMSDOS only retains an active sheet. 36xlTextPrinterFormed text file. Only retains the current active sheet. 42xlUnicodeTextUnicode text file format. Tables (Excel and others) -4143xlWorkbookNormalExcel is the format of the work book file. 39xlExcel5 / xlExcel7Excel versions from 1993 (Excel 5.0) and 1995 (Excel 7.0). The 43xlExcel9795Excel version from 1995 and 1997. However, as author Richard Mansfield explained in VBA development for Microsoft Office 2013, this file format is generally compatible with Excel 95 and later versions. 46xlXMLSpreadsheetXML FILE format. Usually used in Excel 2003. 50xlExcel12Excel 2007. 51xlOpenXMLWorkbook / xlWorkbookDefaultOpen XML workbook / Default Workbook file format. 52xlOpenXMLWorkbookMacroEnabledMacro-Enabled Open XML work book. The 56xlExcel8Excel version is from 1997. File format 60xlOpenDocumentSpreadsheetOpen Document Spreadsheet. OpenDocument Spreadsheet files can be opened using spreadsheet apps that use OpenDocument Spreadsheet. Examples of such apps are Google Sheets, Calc Open Office and Excel itself. Formatting can be affected when open Document Spreadsheet files are saved or opened. 61 (No. H3D)xlOpenXMLStricWorkbookISO Strict format of open XML file. Clipboard Files 44xlHtmlHTML / Web page file format. If you save the Excel work book to a CSV or text file format, two things will happen: Excel selects a code page to use by checking the system's locale configuration on the computer where the work book is stored. The code page used corresponds to the language of the system used. In Windows 10, you can find these settings, go to the settings of the time and language of the region and language. Excel saves the file in a logical layout. This is true, in particular, when working with files containing two-direction text, where the text can be in different directions (left to right and right to left). Whenever text is embedded in one direction in the text in the other direction, the logical layout saves the file in such a way that the order of reading is correct for all languages used, regardless of their direction. Then, when the file is opened later, the entire text in the file (usually) is displayed in the appropriate direction. This is determined by the ranges of characters in the code page used. Let's go back to sample Save_Workbook_NewName. The next screenshot shows what the VBA code of this macro looks like when I add the FileFormat argument and set it up to 52 (Macro-Enabled Open XML workbooks). Argument #3: Password Password Argument Workbook.SaveAs method allows (as you can expect) to enter a password to protect the saved Excel book. Argument Password has the following three main characteristics: This is a line. Sensitive to the case. Its maximum length is 15 characters. The next screenshot shows the VBA code behind the password macro Save_Workbook_NewName. In this case, the password is Excel Tutorial. If you save a work book with a macro such as the one above, the next time someone (you or another user) tries to open an Excel work book, Excel displays a password dialogue. If the wrong password is entered, Excel does not open the work book. Instead, it displays a warning. Argument #4: WriteResPassword Option WriteResPassword method Workbook.SaveAs, in some ways, is similar to the password argument that I will explain above. However, Password and WriteResPassword have one essential characteristic: they protect different things. As explained above, Password protects the work book. If you (or the user) don't provide the correct password, Excel doesn't open the work book. WriteResPassword protects booking records specific to the work book. To see what it is and how it works in practice, I add the WriteResPassword argument to the Save_Workbook_NewName macro. Password for these purposes Excel course. The dialog that Excel displays to ask for WriteResPassword is slightly different from what it uses when requesting a password. Notice how it informs that

the user who saved the work book has reserved it and provides 2 options: You can enter the password and Excel provides you to write access. Otherwise, you can only open your work book for reading. If I choose to open a work book just for reading, Excel does just that. In this case, he warns in several places that the work book is only for reading and changes are not saved. If you type in the wrong WriteResPassword, Excel reacts the same way it does when you enter the wrong password (as shown above). In other words, it doesn't open the work book and displays the following message: Argument #5: ReadOnlyRecommended Argument ReadOnlyRecommended provides you with a less rigorous way (compared to WriteResPassword above) to protect the Excel book you save. More precisely, if you install a specific work book for reading, Excel displays a message, making such a recommendation whenever the file is opened. Installing a working book to read is only recommended not to really protect or work book just like a password or WriteResPassword do. Anyone can only open the recommended Excel Excel book (not so just read), for example: Clicking No in the dialog box above. Installing IgnoreReadOnlyRecommended argument Workbooks.Open argument Truth when opening a book using VBA. To determine that the Excel work book is only recommended for reading, you simply set the ReadOnlyRecom argument to True. Argument #6: CreateBackup CreateBackup argument Workbook.SaveAs method allows you to determine whether backup work book is saved created. If you want to back up the saved Excel work book, set the CreateBackup argument to True. The #7 argument: AccessMode The AccessMode argument allows you to specify access mode for a saved work book. This argument can take the following 3 values: 1: Stands for xlNoChange. In this case, the default access mode is used. 2: Represents xlShared. In this case, the access mode is a list of general data. 3: Value for xlExclusive. In this scenario, the access mode is an exclusive mode. The following screenshot shows the VBA macro code Save_Workbook_NewName with AccessMode installed for xlNoChange: Argument #8: ConflictResolution ConflictResolution is used when working on shared workbooks. More precisely, this argument allows you to determine how conflicts are resolved (while saving the Excel work book). ConflictResolution can be set at any of the following 3 values: 1: Stands for xlUserResolution. In this case, Excel displays a dialog window asking for a resolution to the conflict. This is the default option if you omit the ConflictResolution argument. 2: Represents xlLocalSessionChanges. If you choose this value, changes made by the local user are always accepted. 3: Value for xlOtherSessionChanges. This is the opposite of the above: changes made by a local user are always rejected. The following screenshot shows the macro code Save_Workbook_NewName ConflictResolution, the default xlUserResolution. Argument #9: AddToMru MRU means only recently used. This makes a link to the Excel list of the most recently used files that you usually find on Backstage View. The AddToMru workbook.Save argument allows you to determine if a saved work book has been added to this last list used. If AddToMru is set up for True, Excel is added to the list. AddToMru's default, however, is false. In the following image, you can see the VBA code behind the example of the Save_Workbook_NewName macro with AddToMru, set on True: As mentioned above, I do not cover in detail the arguments of TextCodePage and TextVisualLayout (arguments #10 and #11). The argument #12: The Local's Last Argument Workbook.SaveAs method is local. As the name suggests, Local relates to the language and aspects of localization working book. More precisely, the local setting allows you to determine, determine, The work book is stored in the language: Excel, as is usually defined from the control panel settings. or VBA, which is usually U.S.-English. The main exception to this VBA rule is us-English when the VBA project that runs the Workbook.SaveAs method is an internationalized XL5/95 VBA project. I think you are unlikely to work with such projects often. To determine how Excel works in relation to this topic, you can set a local True or False argument. True: Saves the work book from the Excel language. False: Saves Excel workbook from VBA language. In the following image you can see a sample Save_Workbook_NewName with a local setting installed on True: How to Save A Copy A Copy Of A Excel Workbook Using The Workbook.SaveCopyAs Method VBA Conservation method, as explained above, are the main methods that you need to save Excel books using VBA. However, both of these methods save and change the current open Excel book. You may encounter some situations when this is not the result you want. In other words, you'll probably be in situations where you want a macro simple: Keep a copy of the current Excel book, but... Don't actually change the current file in your computer's memory. This type of situation is great for using Workbook.SaveCopyAs VBA method. This method does just that. He takes the work book and: Keeps a copy of the file. Doesn't change it in memory. The Syntax of saveCopyAs method, again, is relatively simple: expression. SaveCopyAs (Filename) As well as other methods explored in this Excel tutorial, the expression is a Workbook object. Filename, the only option for saveCopyAs is the full file path, name and extension of the copy you save. Since you'll probably use this method on an active work book most of the time, you'll probably end up using the following syntax often: ActiveWorkbook.SaveCopyAs (Filename) Another widely used alternative is to use ThisWorkbook property instead of ActiveWorkbook. The main difference between ThisWorkbook and ActiveWorkbook is that: ActiveWorkbook refers to the current active work book. ThisWorkbook refers to a work book where the macro is actually stored. Consider the macro example that Workbook.SaveCopyAs uses to save a copy of the current active workbook: the screenshot below shows a macro called Save_Copy_Workbook. This macro has one (pretty long) statement. This goes like this: ActiveWorkbook.SaveCopyAs Filename: ActiveWorkbook.Path - Copy and Format (Now, yy-mm-dd) - ActiveWorkbook.Name Notice that the structure I use in the macro Save_Copy_Workbook follows the main syntax Workbook.SaveCopyAs, explained above. However, let's divide the statement into two parts to better understand what this particular method can do for you: Part of the #1: ActiveWorkbook.SaveCopyAs Is a reference to the SaveCopyAs method. This follows from the basic syntax, explained above. ActiveWorkbook refers to application.Workbook. This property returns the Workbook object, which represents the current active work book. This active work book is one that is manipulated by the SaveCopyAs method. In other words, the statement simply says that Excel should act as follows: step #1: Take the current active work book. Step #2: Save a copy of the current active workbook without actually changing it in memory. Part of #2: Filename: ActiveWorkbook.Path, ActiveWorkbook.Path, , , , , , , , , , , , ActiveWorkbook.Name This part of the statement defines the only argument of the workbook. SaveCopyAs method: Filename. This particular file name for a copy is a bit long, but mostly constructed by concatenating 5 elements. You use the ampersand (l) operator to concate various elements. Item #1: ActiveWorkbook.Path This makes a reference to the Workbook.Path property. The Path property returns the full path to the relevant work book. In the example above, ActiveWorkbook.Path is used to get a way to the current active work book. Suppose, for example, that the current active work book (called Book1) is stored in D. In this case, the path is simply D:. This path sample (D:) not very long or difficult. However, in practice, you are more likely to work with longer and more complex ways that you should only work with a D drive. Elements #2 and #4: Copy and It's just text lines. The first line indicates that the first word in the file title is Copy. The second line adds space. Item #3: Format (Now, yy-mm-dd) This specific statement uses 2 built-in VBA features, namely: Now returns today's date and current time. You can also use the Date feature, which returns the current date. The format accepts the date returned to Now and formats it according to the yy-mm-dd date format. In other words, this part of the argument is responsible for returning the date in which the copy is stored in yy-mm-dd format. For example, if the date of saving a copy of a work book is November 30, 2015, this item returns 15-11-30. Item #5: ActiveWorkbook.Name This item uses Workbook.Name property to get the title of the work book. For example, if the title of the working book is Best Excel Tutorial, it Workbook.Name just that. In order to make the all clear regarding the Workbook.SaveCopyAs method, let's look at an example: How to keep a copy of Excel Workbook Using Workbook.SaveCopyAs VBA Method: Example Let's assume that the current active book is called Best Excel Tutorial and Saved in Drive D (D:). Here'Save_Copy_Workbook s what drive D looks like before you run a sample Save_Copy_Workbook Save_Copy_Workbook macro: The next screenshot shows how the same drive looks after the macro launch. yy-mm-dd) - ActiveWorkbook.Name Notice how each of the 5 elements explained above expresses itself in practice after the macro launch: Item #1: The copy is stored in the same folder as the original work book as stated in the properties Workbook.Path. Elements #2 and #4: The first word in the actual title of the work book is Copy, defined by the line Copy. As indicated. Item #3: The date in which the work book is saved (November 19, 2015 in the example above) is added to the name in the yy-mm-dd format (15-11-19). Item #5: The title of the original work book (Best Excel tutorial) is added at the end of the copy name. The following image shows the following: How to name a work book using Application.GetSaveAsFilename I introduced the Application.GetSaveAsFilename method above. This method is used by one example of macros (Save_Workbook_NewName) for the purpose of opening the Save As dialog window and allows users to easily view and enter the path, name, and extension of the Excel stored work book file. The screenshot below shows the VBA code Save_Workbook_NewName macro. Please note that application.GetSaveAsFilename is available. Application.GetSaveAsFilename doesn't actually save the file. However, GetSaveAsFilename is a useful method to use whenever you have a macro that needs to get a file name from the user to, among other things, save the work book. GetSaveAsFilename is useful when the procedure needs to get/know the name of the file to save. This gives the user the ability to specify the file path and file name. As I'll explain below, you can use Application.GetSaveAsFilename for this very purpose. The GetSaveAsFilename method has several options that allow you to customize some of its characteristics. Let's take a closer look at the method itself and its arguments, starting with: Application.GetSaveAsFilename: The purpose of the Application.GetSaveAsFilename method does 2 things: Displays save like a dialog box. Receives the name of the file entered by the user in the Save As dialog window. GetSaveAsFilename does not save a work book by itself. That's why, for example, the Save_Workbook_NewName macro involves using the Workbook.SaveAs method to save the Excel work book. Syntax Full syntax of Application.GetSaveAsFilename is this: expression. GetSaveAsFilename (InitialFilename, FileFilter, FilterIndex, Title, ButtonText) is used to represent the app object. So you probably usually use the following basic syntax for this method: Application.GetSaveAsFilename This is the syntax used in the version of the Save_Workbook_NewName method shown above. All 5 GetSaveAsFilename arguments are optional. Consider them: Application.GetSaveAsFilename: Arguments The following table contains a basic description of the 5 parameters of the Application.GetSaveAsFilename method. I explain each of them more carefully below. PositionNameDescription 1InitialFilenameSpecifies is offered/file name by default. 2FileFilterDetermines file filtering criteria. 3FilterIndexDetermines filter the default file. 4TitleDetermines name (usually called) Save as a dialog box. 5ButtonTextA is only used on the Mac platform. Defines the text of the button (usually called) Save As. There are quite a few similarities between GetSaveAsFilename and GetOpenFilename (which I describe here). In terms of their arguments, the main differences are this: GetSaveAsFilename has an argument. GetOpenFilename does not. GetOpenFilename has the MultiSelect argument. GetSaveAsFilename does not. Both of these differences make sense. For example, MultiSelect allows you to determine whether a user can choose multiple file names at the same time. This makes sense in the context of opening files. But not in the context of saving files using the GetSaveAsFilename method. Consider each of the parameters above: Argument #1: InitialFilename The original name of the Application.GetSaveAsFilename allows you to establish the proposed file name. This proposed file name is the name that appears in the Save As file name box by default. The Save As dialog box, displayed above, is the result of the launch of the next version of the macro Save_Workbook_NewName. Note that the InitialFilename argument is added and the suggested name is Best Excel Tutorial, as shown in the image above. Argument #2: FileFilter Argument FileFilter Application.GetSaveAsFilename allows you to determine the criteria for filtering files in the Save As dialog field. These file filtering criteria define what is displayed in Save as the type of fall box the box is saved as a dialog box. If you omit the FileFilter argument, by default (as shown below) all files. This is not ideal because it can result in a saved Excel work book being unrecognizable if the user does not enter the file extension while saving the file. However, I think you'll be in situations where File filtering criteria are more convenient or even necessary. In order to be able to determine which file filters appear in the Save As dialog field, you will need to follow the 4 guidelines below. Don't worry if the guidelines don't seem that clear at first glance. I'll show you a practical example of VBA code after the introduction and basic description. The guide #1: each filter consists of a pair of lines. Each filter you specify when using the FileFilter argument consists of two lines separated by a comma. It looks, roughly speaking, as follows: String1,String2 String1 and String2 have different structures and goals. More precisely: String1: Is a narrative line. This line defines what actually appears in Save as the type of fall box to save as a dialog box. String2: Is MS-DOS a wildcard file-type filter specification. In other words, this line determines how the files are actually filtered depending on the file format. You don't have to follow many guidelines on how the first line (String1) is listed. However, when you specify the second line (String2), you need to follow a more specific syntax. Let's take a look at it: A guide to #2: Syntax To indicate a filter-type file. The second line used to specify the file filter itself consists of 3 elements, which are usually the following: Element #1: The asterisk (l) is used as a wildcard. Element #2: dot (.). The #3 element: an indication of the file extension used to filter the files. This particular item usually consists of (if necessary) an asterisk (me) used as a wildcard and/or (or (if necessary) a text. The main filter is all the files, which in practice means no filter. To specify a filter-type file than includes all the files using the syntax above, you would enter the point asterisk (me). Other examples of filter specifications for file type after this syntax are: No.txt for text files. V.hla for add-ons. No.xlsx for Excel work books. No.xlsm for macro-Enable Excel workbooks. No.xls for excel 97 to Excel 2003 workbooks. V.csv for CSV files. Knowing these first 2 guidelines is enough for you to start using the FileFilter argument. However, they only explain how to specify one filter according to one type of file. However, when you work with FileFilter, you can specify: several different filters; and several different types of files for each filter. The following two guides show how you can do each one: A guide #3: Syntax To indicate multiple filters. You can create more than one filter with the FileFilter argument. To do this, use commas to separate the filters. In other words, to separate of a pair of lines that make up the filter from another pair of strings using commas .. This looks, roughly speaking, as follows: The #4 guide: syntax to indicate multiple types of files in a single filter. If you need to filter according to several different types of data, you can use multiple filters using the syntax explained above. You can also specify several types of data for a particular filter. To do this, divide the MS-DOS wildcard expressions that are used with commas (;). It looks something like this: String1,String2.1; String2.2 These are the 4 basic guidelines that you need to keep in mind to start using the FileFilter argument. Let's go back to the macros Save_Workbook_NewName create some file filters: the next screenshot shows (again) the VBA code for Save_Workbook_NewName. Note that the FileFilter argument has been inserted and its syntax follows all the guidelines that I explained above. To make this clearer, let's break the meaning of the argument into its various parts and emphasize how it fits all the guidelines described above. The full argument is this: Excel Workbook,.xlsx, Excel Macro-enabled workbook, xlsm, excel templates, .xltx;'. xltm Note the following things: There are 3 filters. Each of the filters is separated from the other by commas. Each filter consists of two parts: a narrative line and an appropriate MS-DOS wildcard filter specification. These two parts are separated by commas.) The MS-DOS filter filter specifications follow the syntax described above: i) an asterisk (l); (ii) point (.); and (iii) file extension specification, without wildcards in this case. The latest filter uses two different types of files. These file types are separated by a goal (;). The following image shows how all of the above looks #3 in practice. This is the first filter that has been listed with fileFilter argument. However, you can change the criteria for filtering files by default using the FilterIndex argument. than the number of filters available (4 or higher in the case of macro Save_Workbook_NewName), the first filter is used. In other words, the practical result of specifying too high a number of indexes is the same as when FilterIndex. The next screenshot shows the code FilterIndex is set at 2. In the case of this macro, FilterIndex 2 means that Excel Macro-Enabled Workbook is the new default filter. Argument #4: Title of argument application. GetSaveAsFilename method allows you to change the name (usually called) Save as a dialog box. If you omit the argument, the default name (Save As) is saved. The following image shows how this argument can be used to change the name of the Save As dialog window when performing a macro Save_Workbook_NewName. In this case, the Title argument is set on the VBA Save Excel Workbook. When you run this macro (formerly called) Save As dialog, here's how it looks. Note that the name has really changed on VBA Save Excel Workbook. The #5 argument: ButtonText ButtonText only applies to the Mac platform. If you use this argument in Windows, it's just ignored. Where this is applicable, the ButtonText argument allows you to set the text that appears in the (usually known as) Save button. Finding knowing how to keep Excel workbooks using VBA is essential. If you've read this Excel tutorial, you now know the basics of how to save workbooks with VBA. In fact, you've seen 3 different ways to achieve this: Using a workbook. Save the method. Using Workbook.SaveAs. Use Workbook.SaveCopyAs. Each of these cases is explained by a real example of VBA code. Also, in the last section of this blog, I explained application.GetSaveAsFilename method. Although this method doesn't actually save the file by itself, it allows you to display save both dialogue so that users of your macro can easily point the way and title of the work book file they save. Save. excel vba activeworkbook.save as read only. excel vba activeworkbook save copy as