Excel VBA: Working with Excel Elements

Excel VBA programming - Part 2

Harun Kaplan

bookboon
The eBook company

HARUN KAPLAN

EXCEL VBA: WORKING WITH EXCEL ELEMENTS

EXCEL VBA PROGRAMMING - PART 2

Excel VBA: Working with Excel Elements: Excel VBA programming - Part 2

1st edition

© 2018 Harun Kaplan & bookboon.com

ISBN 978-87-403-2411-2

Download free eBooks at bookboon.com

CONTENTS

	Introduction	**5**
1	**Programming**	**7**
1.1	Event Procedure	7
1.2	Handling Workbooks	11
1.3	Worksheets	19
1.4	Handling with "Range" and "Cells"	29
1.5	Handling "SpecialCells"	53
1.6	Handling Comments	55
1.7	Setting of cell	58
1.8	Offset (new position) of cursor	65
1.9	Created a link	67
1.10	Setting the best rows wide and best column height	68
1.11	Defining a print area, header, and footer	69
1.12	Automatic filtering	73
	Bibliography	**76**

CMO INSPIRED CONFERENCE
25 OCTOBER | DE VERE BEAUMONT ESTATE | OLD WINDSOR UK
Join Over 100 Chief Marketing Officers & Digital Innovators
Click on the ad to read more
Download free eBooks at bookboon.com

INTRODUCTION

So far we have learned:

In the first book "Excel-VBA Introduction": General Information, Editor Environment and Language Concept of VBA Programming;

In the second book "Excel-VBA Working with Excel Elements":

- Workbooks "Workbooks"
- Sheets "Worksheets"
- handling cells and ranges "ranges" / "cells"
- dealing with the "SpecialCells" method
- insert function / formula
- Handling comments

The target group for the book includes beginners as well as advanced users

Harun Kaplan

Download free eBooks at bookboon.com

For my Family:
Tülay
Yasin, Sueda, Melik

Download free eBooks at bookboon.com

1 PROGRAMMING

If we look at VBA programming as a body, the topics so far were its hands and feet. From now on, we will focus on the guts of the VBA body. This part is extensive but also exciting. We are accompanied from one topic to the next.

1.1 EVENT PROCEDURE

An event is an actual state. That is, a procedure is executed after a certain state is reached. In other words, to be able to execute a procedure, a certain actual state must apply. Each file and every table are treated as objects in event-oriented programming language.

The most important object events are:

- Workbook opens all events of a file. By double-clicking on the "This workbook" in the project window, we reach its area. Then we open the appropriate event procedure. For example, "Open"; "BeforeClose"; "BeforeSave"; "Activate".
- Worksheet opens all events of a table. By double-clicking on the desired table in the project window, we reach its area. For example, "SelectionChange", "Activate"; "Change".

The names of the event procedures are fixed or rigidly defined.

This consists of:

- Object names
- Underscore "_" and
- Event Names

For example: "Workbook_open" or "Worksheet_Change".

1.1.1 EVENTS OF WORKBOOK_

All "Workbook" events are listed in the table below. We will discuss "Open" and "BeforeClose" here.
The first example is with "open": When we open our Excel file, it should greet me with "hello colleague". Our procedure with the matching event, "open", looks like this:

Download free eBooks at bookboon.com

To insert an open event, proceed as follows:

- Call up Project Explorer for VBA development,
- Double-click on the "This workbook" to get to the VBA editor,
- From the left menu, select the entry "Workbook" and in the right-hand menu select "open"
- Then we write our procedure

```
Private Sub Workbook_Open()
 Worksheets("VBA").Select
 Range("A1").Select
 MsgBox "Hello Colleague"
End Sub
```

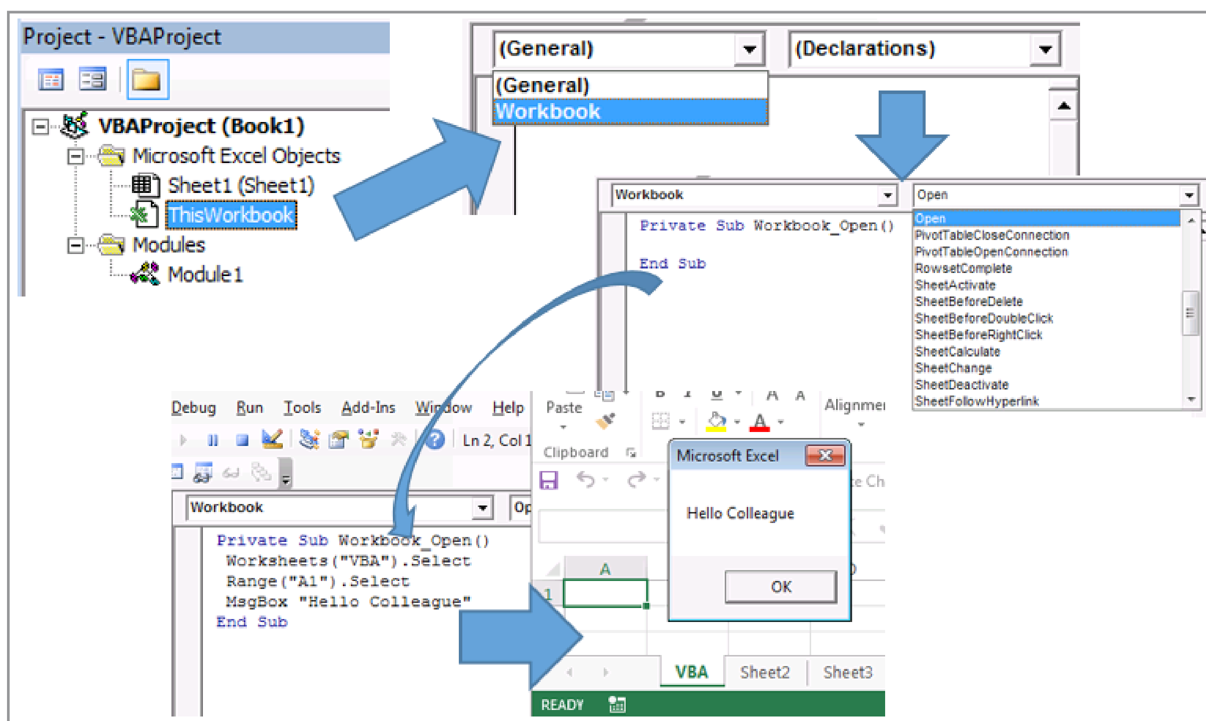


Figure 1: Create workbooks_open event

Next example with "open": When we have opened the Excel file, the following actions are performed:

- The table "Database" preselected,
- The command "**Call**" executes "Calling_Menu",
- Using .**OnKey** "Diagram creation" is assigned to the "**F9**" function key
- With "**OnTime TimeValue**", the "Database_update" time specified is implemented.

```
Private Sub Workbook_Open()
  Worksheets("Datebase").Select
  open Calling_Menu
  Application.OnKey "{F9}", "Create a Chart"
  Application.OnTime TimeValue("08:01:00"), "Datebase_update"
End Sub
```

With "BeforeClose" VBA executes specified actions before exiting the Excel file. An example with "BeforeClose": In the previous example we listed "Calling_Menu". The content is reset when the Excel file is closed. Detailed information will be discussed later in the chapter "Own Menu Creation".

```
Private Sub Workbook_BeforeClose()
  Application.CommandBars("Worksheet Menu Bar").Controls("Datebase open").Delete
  Application.CommandBars("Worksheet Menu Bar").Controls("Activate").Delete
End Sub
```

The lower table lists important events of the object „Workbook".

End Sub Events of object Workbook	
Activate	Executed after changing the excel file
BeforeClose	Executed before exiting the excel file
BeforePrint	Executed before printing the excel file
BeforeSave	Executed before saving the excel file
Deactivate	Executed after changing the excel file
NewSheet	Executed after adding a new excel sheet
Open	Executed when opening the excel file
SheetActivate	Executed when a sheet is changed (sheet is activated)
SheetBeforeDobleClick	Executed when double-click on the respective sheet
SheetCalculate	Executed when references or formulas are recalculated
SheetChange	Executed when at least changed to a cell

Table 1: Events of Workbook-Object

1.1.2 EVENTS OF WORKSHEET_

A worksheet event is defined by the worksheet object. This event is activated when a sheet is activated / deactivated, when a cell is changed, or when the pivot table is activated. Here we proceed exactly as in the previous chapter.

To insert a worksheet event, referred to here as "Activate", proceed as follows:

- Open Project Explorer for VBA development,
- mark the relevant table, here "Sheet2",
- select from the left list box "Worksheet",
- select the desired event from the right, here "Activate".

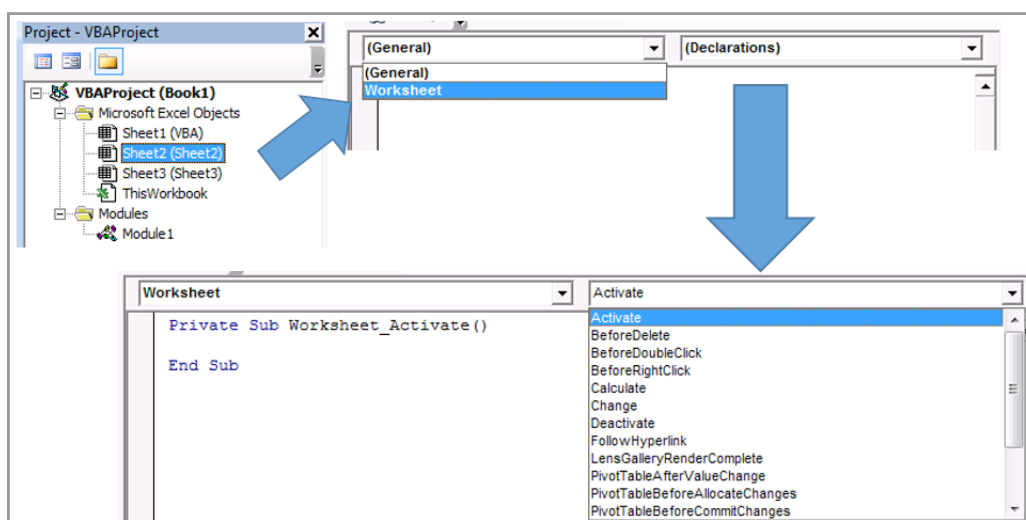


Figure 2: Create a worksheet event

First, an example with "Activate". As soon as the respective table is selected, "Activated", a message should appear:

```
Private Sub Worksheet_Activate()
    MsgBox "I was selected!"
End Sub
```

The second example is to fill the active or selected cell with the color "yellow". However, this only happens in the table "Sheet2". In other tables, this event has no function.

Now for our example: The statements **ByVal Target As Range**", meaning "activities in the cell", are in the hull.

Download free eBooks at bookboon.com

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    With Selection.Interior
        .ColorIndex = 6
        .Pattern = xlSolid
        .PatternColorIndex = xlAutomatic
    End With
End Sub
```

The lower table lists important events of the "Worksheet" object.

Events of object Worksheet	
Activate	Executed after changing a sheet
BeforDobbleClick	Executed when double-click within a sheet
BeforRightClick	Run when right-click inside a sheet
Calculate	Executed when references or formulas are recalculated
Chage	Executed when at least changed to a cell
Deactivate	Executed when switching from the sheet to another
SelectionChange	Executed when the marker in sheet changes

Table 2: Events of Worksheet-Object

1.2 HANDLING WORKBOOKS

We recognize an excel file at its xls * extension. These Excel files are also called "workbook".

When being declared, they are defined as a workbook object.
For example: Dim <variable name> As Workbook.

We will cover the topic "Workbooks" in the following sections:

- .Add
- .Open
- .Save / .SaveAs / .Saved
- .Closed
- .Activate

Each method may have a long list of factors to consider. In the following examples we will look at only the most important points for each method. More detailed information can be read in the help text. Position the cursor on the method, then press the “F1” key.

1.2.1 CREATE A WORKBOOK

An empty workbook is created temporarily with “**Workbooks.Add**”. It should be saved before exiting Excel. If it was not saved in a specified location, it will be stored in the “default” (my documents”) directory.

This directory can be checked under menu “**Tools | Option**” in the tab “**General**” or in a specified new directory.

The extension .xls* does not have to be specified explicitly because it is attached automatically when saving.

Here we can see a few examples of how VBA creates this type of workbook:

```
Sub Create_a_workbook_without_save()
  Workbooks.Add
End Sub

Sub Create_a_workbook_with_save_1 ()
  Workbooks.Add.SaveAs ("Harun_Test2.xlsx") 'saved in default directory
End Sub

Sub Create_a_workbook_with_save_2()
  Workbooks.Add.SaveAs ("C:\Excel\Harun_Test.xlsx") 'saved in "C:\Excel"
End Sub

Sub Create_a_workbook_SET()
  Dim NewWorkbook As Workbook
  Set NewWorkbook = Workbooks.Add
  'NewWorkbook.SaveAs ("Harun_Test.xlsx") 'saved in default directory
  NewWorkbook.SaveAs ("C:\Excel\Harun_Test.xlsx") 'saved in "C:\Excel"
End Sub
```

With the SET statement we can simplify our work a bit. This assigns a variable to the statement. This variable, here “wbNewMap”, should be dimensioned beforehand with the DIM statement as a workbook.

The example below is in the help of the SaveAs in VBA. This procedure creates a new workbook and prompts the user for a file name. Then it will be saved under new names.

```
Sub NewWorkbook_saved_as()
 Set NewWorkbook = Workbooks.Add
 Do
    New_Name = Application.GetSaveAsFilename
    Loop Until New_Name <> False
    NewWorkbook.SaveAs Filename:=New_Name & ".xlsx"
End Sub
```

1.2.2 DETERMINE A WORKBOOK

The file name, or filename with the path, can be determined using these two properties:

- **ThisWorkbook** => the file from which the procedure was executed,
- **ActiveWorkbook** => the file that is in the active window (the window in the foreground).

Filename ONLY: <*Expression*>**.Name**
Filename with sub: <*Expression*>**.FullName** or **.FullNameURLEncoded**

Our file is called Workbook.xlsx. Our example "Workbook" is also here. Now we call a second file "File_ThisWorkbook.xlsx". Then let's take our example:

```
Sub Example_Workbook()
Dim strName, strFullname, strFullnameURL As String
Dim strName2, strFullname2, strFullnameURL2 As String

strName = ThisWorkbook.Name
strFullname = ThisWorkbook.FullName
strFullnameURL = ThisWorkbook.FullNameURLEncoded

strName2 = ActiveWorkbook.Name
strFullname2 = ActiveWorkbook.FullName
strFullnameURL2 = ActiveWorkbook.FullNameURLEncoded
MsgBox "Macro executed from the file: " & _
          vbLf & strName & vbLf & strFullname & vbLf & strFullnameURL & vbLf & _
          "Active file is " & vbLf & strName2 & vbLf & strFullname2 & vbLf & strFullnameURL2
End Sub
```

Download free eBooks at bookboon.com

Our result looks like this:

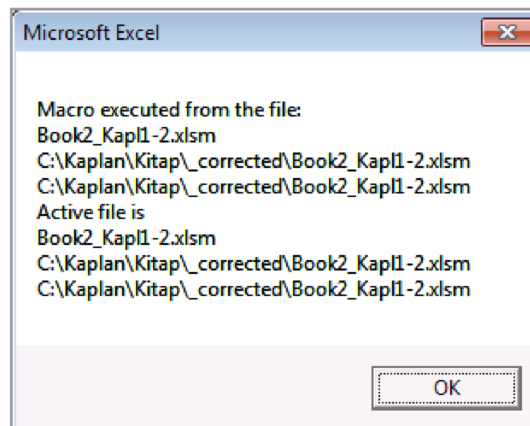


Figure 3: Information of same workbooks in MsgBox

A result with a server specification may look like this:

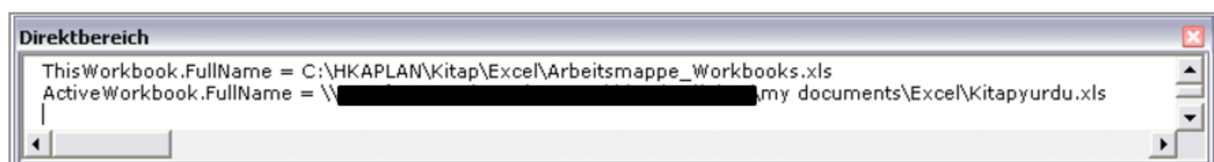


Figure 4: Example Fullname of a file

1.2.3 OPEN A WORKBOOK

To open a workbook use **Workbooks.Open**.
If we did not specify a specific directory, it will be searched for in the default location.

```
Workbooks.Open ("Harun_Test.xlsx") 'opened from the default location
```

If not foundnd, an error message will be displayed.

For example, a full directory path looks like this:

```
Workbooks.Open ("C:\Excel\Harun_Test.xlsx") 'opened from the directory "C:\Excel"
```

If any of the cells in the table have links, they can be turned on or off for refresh. This is switched on with "**UpdateLinks: = 1**" or switched off with "**UpdateLinks: = 0**".

```
Workbooks.Open ("C:\Excel\Harun_Test.xlsx", UpdateLinks:=0, ReadOnly:=True) 'opened from the directory "C:\Excel", without updating the links, opened as read-only
```

With a password, it looks like this:

```
Workbooks.Open ("C:\Excel\Harun_Test.xlsx", Password:="Harun") 'The password-protected folder from the directory "C:\Excel" is released for writing.
```

We can also choose a workbook by opening dialog. Then our example may look like this:

```
Sub Dialog_File_Open_1() 'from default location
Dim strDatei As String
'Stating about menu File | Open
   strDatei = Application.GetOpenFilename("My Excel files (*.xls*), *.xls*")
'Starting with selection
   Workbooks.Open Filename:=strDatei
End Sub

Sub Starting_specified_directory() 'Start from the specified directory
Dim strZiel, strPath, strFile As String
'Select Path
strPath = "C:\KAPLAN"
ChDrive "C"
ChDir strPath
'Start File | Open window
   strFile = Application.GetOpenFilename("My Excel files (*.xls*), *.xls*")
'Start the selected file
   Workbooks.Open Filename:=strDatei
End Sub
```

1.2.4 SAVE A WORKBOOK

Closing a workbook is the responsibility of **Workbooks.Save**. These are the following properties and methods for saving the workbooks:

- SaveAs This method saves the workbook under a new name
- Save This method saves changes in the specified workbook
- Saved This property checks if a workbook has been saved since the last change.

```
Sub Save_with_SaveAs_1() 'saved the active workbook under a new name
ActiveWorkbook.SaveAs ("VBA.xlsx")
End Sub

Sub save_mit_save_1() 'saved the active workbook only
  ActiveWorkbook.Save
End Sub

Sub save_mit_save_2() 'save all open workbooks
Dim AMappen As Workbook
  For Each AMappen In Application.Workbooks
    AMappen.Save
  Next AMappen
End Sub
```

The next example saves the file under new names if it has not been saved since the last change.

```
Sub Saved_with_SaveAs_3()
  If ActiveWorkbook.Saved Then
    MsgBox "Already saved."
  Else
    ActiveWorkbook.SaveAs ("Harun_TTest.xlsx")
  End If
End Sub
```

1.2.5 CLOSE A WORKBOOK

Closing a workbook is the responsibility of **Workbooks.Close**. Everything that was created or opened must be closed again. The method .Close closes a workbook. If we quit without attributes, for example by using just the Close method, and have saved a change before, then a "Save" message will be displayed.

```
Workbooks("Harun_Test.xlsx").Close 'Workbook will be closed

ActiveWorkbook.Close 'The active workbook will be closed

ThisWorkbook.Close SaveChage:=True ' The active workbook will be closed with saving.

Workbooks("Harun_Test.xlsx").Close SaveChanges:=True 'Closed with saving
Workbooks("Harun_Test.xlsx").Close SaveChanges:=False 'Closed without saving
```

The following example ends all workbooks and saves the changes made to them except the workbook from which the procedure is run.

```
Sub Close_all_workbooks ()
  For Each wb In Workbooks
        If wb.Name <> ThisWorkbook.Name Then
              wb.Close SaveChange:=True
        End If
  Next wb
End Sub
```

1.2.6 ENTER DATA IN THE CLOSED FILE

With the two previous examples, we will write a procedure so that some entries in a “closed” file are entered. In practice, the target file is opened in the background and values to be entered are entered and saved.

Exact handling of Cells or Range will be discussed in more detail later.

```
Sub WriteinanExcelfile()
  Dim strName, strFirstname As String
  Dim intCell As Integer
    'Deactivate of screen updating
    Application.ScreenUpdating = False
      'Target file opens in the background
      Workbooks.Open Filename:="C:\Excel\Test_Harun.xls"
        'Find the last writable cell in the target file
        intZelle = Cells(Rows.Count, 1).End(xlUp).Row + 1
          'entries to be entered
          strName = "Kaplan"
          strVorname = "Harun"
        'Entry
        Cells(intCell, 1).Value = strName
        Cells(intCell, 2).Value = strFirstname
      'Target file closed with save
      ActiveWorkbook.Close SaveChanges:=True
    'Activate of screen updating
    Application.ScreenUpdating = True
End Sub
```

I think our example is self-explanatory. We can continue.

1.2.7 DETERMINE THE NUMBER OF OPEN FILES

With the function “**Workbooks.Count**” we can determine the number of open workbooks. Our example:

```
Sub Determine_Workbooks_Count()
Dim intCount As Integer
        intCount = "Count of open workbooks: " & Workbooks.Count
        MsgBox intCount
End Sub
```

Download free eBooks at bookboon.com

1.2.8 PROTECTED A WORKBOOK OR A WORKSHEET

Of course, a workbook or a table can be protected and the protection can be eliminated. With the "**Protect**" method, a workbook / worksheet can be protected from being changed or the protection can be eliminated with the "**Unprotect**" method.

The example looks like this:

```
Sub Protected_and_unprotected()
Dim strFilename As String
 strFilename = ActiveWorkbook.Name
 Workbooks(strFilename).Protect 'Workbook protect is active
 Workbooks(strFilename).Unprotect ' Workbook protect is deactive
End Sub
```

1.2.9 PROTECTED A CELL OR RANGE

Certain cells can be protected if necessary with the "**.Loced**" method. The first example protects the "C2: D5" area of a table with the "**.Loced = True**" method. Before doing so, we remove protection from the current worksheet using the "**.Cells.Loced = False**" method.

```
Sub Protected_specific_area()
    ActiveSheet.Unprotect "Harun"
    ActiveSheet.Cells.Locked = False
    ActiveSheet.Range("C2:D5").Locked = True
    ActiveSheet.Protect "Harun"
End Sub
```

In the next example, we see the reverse of the previous example. That is, the whole sheet is protected with the ".Cells.Loced = True" method, and certain areas are unprotected with the ".Loced = False" method.

```
Sub Unprotected_specific_area()
    ActiveSheet.Unprotect "Harun"
    ActiveSheet.Cells.Locked = True
    ActiveSheet.Range("C2:D5").Locked = False
    ActiveSheet.Protect "Harun"
End Sub
```

Download free eBooks at bookboon.com

1.3 WORKSHEETS

In the following chapter we will discuss the possibilities of accessing worksheets via VBA code and adjusting them to our wishes.

1.3.1 SELECTION OF WORKSHEETS

There are two alternatives in the VBA for selecting a worksheet:

- Selection via the index of the worksheet object Table Sheet
- Selection by the name of the worksheet object

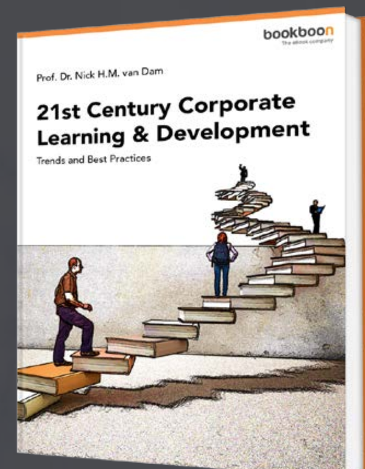
There are three ways to address the names of the worksheets:

1. **Address directly:**
    - Worksheets("Sheet1").Activate

Free eBook on
Learning & Development

By the Chief Learning Officer of McKinsey

Download Now



Click on the ad to read more

2. **Direct address with exact information:**
   This can possibly generate long instructions:
   - Worksheets("Sheet1").Cells(3,5).Value = 233
   - Worksheets("Sheet1").Range("E3").Value = 233

3. **SET-Definition:**

```
Sub Setting_with_SET()
 Dim wsSpreadsheetName as Worksheet
   Set wsSpreadsheetName = Worksheets("Sheet1")
    wsSpreadsheetName.Cells(3,5).Value = 123
End Sub
```

In a workbook the Excel file can open multiple spreadsheets. Therefore, they are entered with the VBA method "Worksheets", that is, plural and defined for declaration as a worksheet object.

We will discuss the topic "worksheets" in the following sections.

- .Add
- .Activate / .Select
- .Copy / .Move
- .Print / .PrintPreview
- .Delete
- .Visible

1.3.2 CREATE A WORKSHEET

With the "**Worksheets.Add**" method a new worksheet can be inserted. The syntax of the Add method looks like this with its attributes:

```
Sub adding_a_worksheet()
worksheets.Add
End Sub     Add([Before], [After], [Count], [Type]) As Object
```

Figure 5: Attribute the Add-Method

In our next example, the cells of "Sheet1" are filled with the following information: A15 = apple, A16 = melon and A18 = pear.

Now we want to create three spreadsheets named "Apple, Melon and Pear" in front of the "Books" table. The generated table sheet names are each assigned a background color, here they appear in red. Now our listing looks like this:

```
Sub Adding_a_worksheet()
Dim x As Integer
Dim Name As Range
For x = 1 To 3
  Set Name = Worksheets("Sheet1").Range("A" & 14 + x)
  Worksheets.Add Before:=Sheets("Books")  'Adding 3 sheets before sheets "Books"
  ActiveSheet.Name = Name.Value ' Sheet name from cell contents of "Sheet2" from Cells "A14..16"
  ActiveSheet.Tab.ColorIndex = 3 'The background colour of the worksheets is defined. 3 = red
Next x
End Sub
```

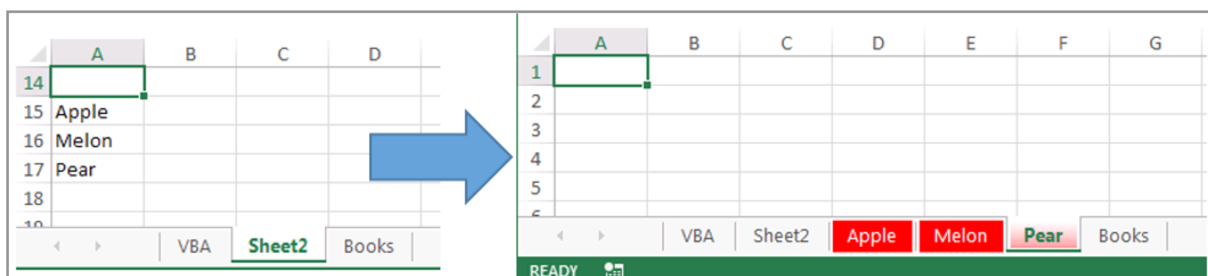


Figure 6: Result of Adding_a_Worksheet

Listing these worksheets can work as follows:

```
Sub Which_worksheets_are_there()
Dim intCount, x As Integer
intCount = Worksheets.Count
For x = 1 To intCount
   Debug.Print Sheets(x).Name
Next x
End Sub
```

The next example shows how to check if a particular table exists. If it does not exist, then it should be created:

```
Sub Check_if_there_is_a_specific_sheet()
Dim wsName As Worksheet
Dim strSearch As String
strSearch="Apple"
For Each wsName In Worksheets
    If wsName.Name = strSearch Then 'Active sheet is compared with the searches sheet
          MsgBox "The sheet " & wsName.Name & " is available."
          Exit Sub
    Else
    End If
Next wsName
  Worksheets.Add Before:=Sheets("Books")
  ActiveSheet.Name = strSearch
End Sub
```

Download free eBooks at bookboon.com

1.3.3 SELECT A WORKSHEET

Using the **Worksheets.Activate** or **Worksheets.Select** methods, we can highlight a table or specify tables to be highlighted. In the current example, the searched table is highlighted after being found. If not found, it will be created.

```
Sub Check_if_there_is_a_specific_sheet()
Dim wsName As Worksheet
Dim strSearch As String
strSearch="Apple"
For Each wsName In Worksheets
   If wsName.Name = strSearch Then 'Active sheet is compared with the searches sheet
        MsgBox "The sheet " & wsName.Name & " is available."
        Worksheets(strGesucht).Select
        Exit Sub
   Else
   End If
Next wsName
 Worksheets.Add Before:=Sheets("Books")
 ActiveSheet.Name = strSearch
End Sub
```

1.3.4 COPY A WORKSHEET

All created worksheets can be copied using the **Worksheets.Copy** or **Sheets.Copy** method. Copying is the cloning of a spreadsheet. This sheet can be placed with the attributes "Before = before specified sheet" or "After = after specified sheet".

After this process, there are two exact same worksheets. If the new worksheet is not renamed, the same term gets parenthesized with an index as a number. For example, "Sheet2" is cloned as "Sheet2 (2)". Therefore always enter a new name immediately.

Now our example looks like this:
We will copy the worksheet "Sheet2" before the sheet "Melon" and then rename it to "New Name".

```
Sub Copy_sheet()
   Sheets("Sheet2").Copy Before:=Sheets("Melone")
   ActiveSheet.Name = "New Name"
End Sub
```

If the copied sheet is placed at the end of the worksheets, we will ask for the number of worksheets.

```
Sub Take_on_last_position()
Dim intCounter As Integer
  intCounter = ThisWorkbook.Sheets.Count 'Ask of count on sheet
  Sheets("Melone").Copy After:=Sheets(intCount) 'The sheet „Melon" is copied and at the end
  ActiveSheet.Name = "New Name" 'Sheet with new name
End Sub
```

It certainly happens that the copied worksheet is not inserted in the same workbook but in a new workbook. This is showed as follows:

```
Sub Copy_sheet()
   Sheets("Sheet2").Copy 'The copied sheet will be added in new workbook
   ActiveSheet.Name = "New Name" 'if necessary, it can also be renamed
End Sub
```

We can also copy multiple spreadsheets at the same time. In our example, three worksheets are copied and pasted before the first worksheet. Thus, they automatically become Sheets (1), Sheets (2), Sheets (3). Then I can easily rename them.

```
Sub Multiple_copy_position_in_the_beginning()
   Sheets(Array("New Name", "Books", "Melon")).Select
   Sheets(Array("New Name", "Books", "Melon")).Copy Before:=Sheets(1)
   Sheets(1).Name = "Template_1"
   Sheets(2).Name = "Template_2"
   Sheets(3).Name = "Template_3"
End Sub
```

What happens when they are added to the end? Not much changes. Below, we are working with the determined number of worksheets:

```
Sub Multiple_copy_position_in_the_ending()
Dim intCount As Integer
intCount = ThisWorkbook.Sheets.Count
   Sheets(Array("New Name", "Books", "Melon")).Select
   Sheets(Array("New Name", "Books", "Melon")).Copy After:=Sheets(intCount)
   Sheets(Amount + 1).Name = "Template_1"
   Sheets(Amount + 2).Name = "Template_2"
   Sheets(Amount + 3).Name = "Template_3"
End Sub
```

1.3.5 MOVE A WORKSHEET

The method **Worksheets.Move** or **Sheets.Move** is responsible for moving a worksheet. Moving is similar to copying a spreadsheet. The difference is that it is "cut out" from the old place and "planted" in a new place.

We can use the same example from Copy and simply replace the Copy method with the Move method.

```
Sub Move_sheet()
  Sheets("Sheet2").Move Before:=Sheets("Melon") '"Sheet2" will move before "Melon"
  ActiveSheet.Name = "New Name" 'if necessary, can also be renamed
End Sub
```

It can also be moved to the end:

```
Sub Move_sheet_position_in_the_ending()
Dim intCount As Integer
intCount = ThisWorkbook.Sheets.Count 'Ask of count on sheet
Sheets("Melon").Move After:=Sheets(intCount) 'The sheet „Melon" move in the ending
End Sub
```

1.3.6 DELETE A WORKSHEET

The sheet is permanently deleted using the **Worksheets.Delete** or **Sheets.Delete** method.

The first example deletes the sheet "Template_1".

```
Sub Delete_sheet()
   Sheets("Template_1").Delete
End Sub
```

It is also possible to delete multiple sheets.

```
Sub Delete_multiple_sheets ()
   Sheets(Array("Template_1", "Template_2", "Template_3")).Select
   ActiveWindow.SelectedSheets.Delete
End Sub
```

1.3.7 SAVE A WORKSHEET AS A FILE

A desired or current worksheet can also be saved as a separate Excel file. Our example:

```
Sub Save_sheet()
Dim strFilename, strPath As String
strFilename = ActiveSheet.Name
strPath = "C:\Kaplan\Kapitel_10\"
ActiveSheet.Copy
ActiveWorkbook.SaveAs Filename:=strPath & strFilename
MsgBox "The file saved in directory " & vbLf & _
       strPath & vbLf & _
       "as" & vbLf & _
       strFilename & ".xlsx"
End Sub
```

Download free eBooks at bookboon.com

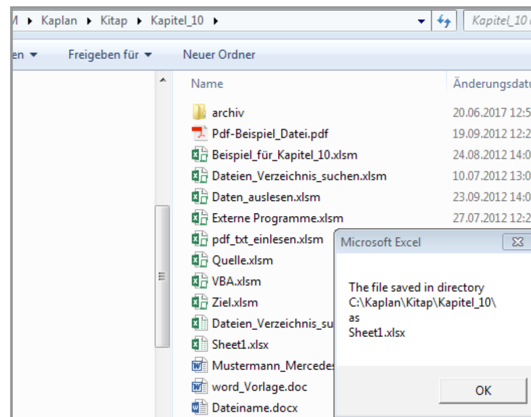


Figure 7: Sheet saved as file

1.3.8 HIDDEN A WORKSHEET

The worksheets can be shown or hidden using the **Worksheets.Hidden** and **Worksheets.Visible** methods. This makes the worksheet invisible from the Excel interface. However, accessing the existing data in hidden worksheets is still possible.

In the example below, the table "Harun" is hidden. Cell "A1" is selected and the output is shown as a MsgBox.

```
Sub Hidden_and_more()
Dim strInhalt As String
Worksheets("Harun").Visible = False 'The sheet "Harun" is hidden
strInhalt = Worksheets("Harun").Range("A1").Value 'Value from "A1" will be selected as String
MsgBox stInhalt 'Selected value will be displayed in a MsgBox
Worksheets("Harun").Visible = True 'The Sheet "Harun" will be visible again
End Sub
```

In the following example, all the worksheets in the active workbook are displayed. Normally, a spreadsheet should continue according to preset functions. When I try to hide all of the worksheets, I get an error message on the last worksheet. To work around this error message, I insert the On Error Resume Next statement. Now our example:

```
Sub All_sheets_hidden()
Dim wsName As Variant
On Error Resume Next 'Error Message will be circumvented
'Hiding all sheets
For Each wsName In Sheets
   wsName.Visible = False
Next wsName
'All sheets will be visible
For Each wsName In Sheets
   wsName.Visible = True
Next wsName
End Sub
```

Now on to something simple. Line “5” and column “F” are hidden and then displayed again.

```
Sub Rows_column_hidden ()
Rows("5:5").Select
 With Selection
  .EntireRow.Hidden = True
  .EntireRow.Hidden = False
 End With

Columns("F:F").Select
 With Selection
 .EntireColumn.Hidden = True
 .EntireColumn.Hidden = False
 End With
End Sub
```

1.3.9 LISTING ALL WORKSHEETS

If we want to list the worksheets, we need the **Worksheets (x)** .Name method.

The next example lists all existing sheets, including hidden sheets. We first get the number of existing sheets with the **Worksheets.Count** property and assign the result to Count. Then we go through “number” times.

```
Sub Which_Spreadsheets_are_there()
Dim intCount, x As Integer
intCount = Worksheets.Count
For x = 1 To intCount
   Debug.Print x & ".Sheet: " & Worksheets(x).Name
Next x
End Sub
```

1.3.10 SETTING BACKGROUND COLOR OF A SHEET NAME

The background color can also be changed as desired using the “**.Tab.Colorindex**” method. If we view the worksheets while turned by 180 °, they look like registers. That’s why “**.Tab**” is used. In order to change the background colors of these tabs, we need “**Color Index**”.

Now an example:

```
Sub Backgroundcolor_of_sheetname()
Sheets("Sheet1").Tab.ColorIndex = 3
End Sub
```

1.3.11 SORTING OF ALL WORKSHEETS

If we have inserted several worksheets, we may need to sort them. To do this, our listing might look like this:

The For-Next loop is great for sorting worksheets. We can sort by ascending or descending order

```
Sub Spreadsheets_Sorting()
   Dim intCount_sheets, x, y As Integer
   intCount_sheets = ActiveWorkbook.Worksheets.Count
   For x = 1 To intCount_sheets
      For y = x To intCount_sheets
         If Worksheets(y).Name < Worksheets(x).Name Then
            Worksheets(y).Move Before:=Worksheets(x)
         End If
      Next y
   Next x
End Sub
```

1.3.12 ON OR OFF CONFIRMATION TO DELETE

Before each deletion of an object (table, row, column, cell) we are questioned for safety's sake. If this query should be too annoying, it can be switched on or off by using the function **.DisplayAlerts**.

Our example is as follows:

```
Sub Delete_sheet_without_query()
   Application.DisplayAlerts = False 'confirmation is off
   Sheets("Template_3").Delete
   Application.DisplayAlerts = True 'again, confirmation is off
End Sub
```

If it is turned off, please turn it back on at the end of the listing. Otherwise, all messages that normally appear on the screen will no longer appear. This can be dangerous under certain circumstances.

Download free eBooks at bookboon.com

1.3.13 SUPPRESS THE MESSAGE "UPDATE LINKS"

If the file to be opened contains at least one link, we will be notified about updates each time it is opened. This message can also be suppressed.

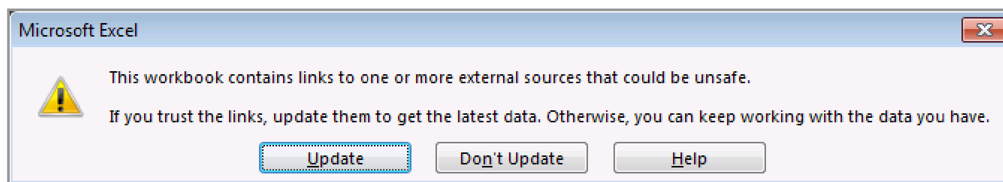


Figure 8: Message of the update query

If we want to open and edit such tables via VBA, the macro process stops at this point. In such cases, we either work with an **AskToUpdateLinks** property or with the parameter of the Open **method UpdateLinks = 0**.

The difference is that the**AskToUpdateLinks** property does the update without asking and the **UpdateLinks = 0** parameter does not.

```
Sub Skip_the_query_AskToUpdateLinks()
  Application.AstToUpdateLinks=False
  Workbooks.Open "C:\Excel\Example_Links.xlsx"
End Sub
```

If a link goes nowhere or if there is an incorrect link, a query appears on the monitor despite the above process.

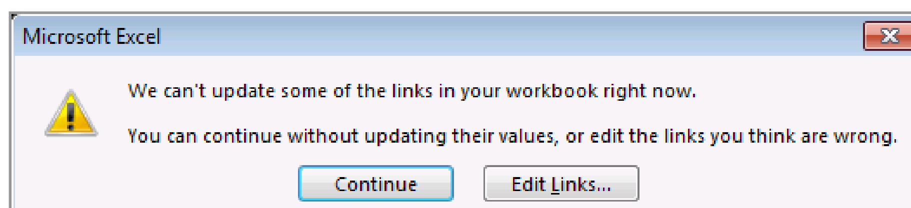


Figure 9: Message at e.g. incorrect link

Our second method works without updating the links. To do this, we'll change the **UpdateLinks** parameter to zero in the Open method:

```
Sub Skip_the_query_AskToUpdateLinks()
  Workbooks.Open "C:\Excel\Example_Links.xlsx", UpdateLinks := 0
End Sub
```

Download free eBooks at bookboon.com

1.4 HANDLING WITH "RANGE" AND "CELLS"

One cell and one cell range in VBA programming covers most of the programming with respect to workbooks "workbooks" or worksheet worksheets. When a cell or area is selected, there are many ways to format it. Marking is a selection or response of a cell or area.

Access to cells or cell ranges is not trivial because VBA works with many terms and objects that are similar in content. As you say, "All roads lead to Rome." Many examples of methods that we will see in this book can be carried out in different ways.

A cell / area is selected for:

- To edit
- Copy
- Cut out
- Insert
  - rename
  - Format to
  - alignment
  - Layout
  - frame design
  - Color Fill
  - protection
  - Show or hide the columns / row
  - Automatic column width and row height

- Sort by
- Set to zero
- Insert formula
- Set pressure range

The cells are addressed with the following options:

- with the Range object
- with the Cells object
- a mix of the two.

The Range object is addressed with the usual cell name and the Cells object is addressed with the cell and column numbers.

The Range object writes a cell inside two parentheses and quotes. The column is written first and then line is written afterwards.

For example, (“A1”) => The letter “A” indicates the column. The number “1” indicates the line.

For the Cells object, a cell is written inside open and closed parentheses and separated with a comma. In this case, the line is written first and the column afterwards.

For example, (3,4) => The first digit (3) indicates the line and the second digit (4) indicates the column-D. This notation is particularly suitable for working with variables.

Range	Cells
(„A1“)	(1,1)
(„D3“)	(3,4)
(„Z13“)	(13,26)

Table 3: Range / Cells

Before we start with the cells and areas, let’s see how the cursor can be positioned in the cell. The positioning is done either with the methods “**Select**” or “**Activate**”.
The Select method marks a cell or range of cells.
The Activate method activates a single cell that is inside the current marker.

In the following example in the table “VBA” the range is marked with Range (“A1: C3”) and with cell (1,1) the cell “A1” is activated so the cursor is in the cell “A1”.

```
Sub Example_Activate_Select()
  Worksheets("VBA").Activate
  Range("A1:C3").Select
  Cells(1,1).Activate
End Sub
```

The following examples set the font in cell A1 of the VBA table to blue.

Now, examples with and without Select:

```
Sub Example_Activate_Select()
  Worksheets("VBA").Activate
  Range("A1").Select
  Selection.Font.Color = vbBlue
End Sub
```

Now, little faster in the execution and less typing:

```
Sub Example_Activate_Select()
  Worksheets("VBA").Range("A1").Font.Color = vbBlue
End Sub
```

If we want to add something to the cell, then we can work WITHOUT Select or Activate method. This makes our procedure a lot shorter and faster. Here is an example:

```
Sub Inputs()
 Text = "Harun"
'
 Range("C1").Activate
 Range("C1").Value = Text
 ActiveCell.Value = Text
 Cells(4, 3).Activate
 Cells(4, 3).Value = Text
 ActiveCell.Value = Text
'
 Range("B1").Select
 ActiveCell.Value = Text
 Cells(3, 3).Select
 ActiveCell.Value = Text
'
 Range("A1") = Text
 Cells(2, 3) = Text
End Sub
```

1.4.1 COMBINE CELLS WITH RANGE

It is also possible to combine both types of writing.

The combination with the Cells and Range is mostly used in loops because the cell ranges are better addressed by means of calculated indices.

The example below fills the following cells with calculated values:

Instead of explaining each line, please type in and run “step-by-step” with “F8”. ☺

```
Sub Combination_Cells_Range()
Dim i As Integer
  For i = 2 To 4
    Worksheets("Tabelle1").Range(Cells(2, 2), Cells(i + 2, i)).Value = 20 * i
  Next
End Sub
```

Or we mark a specific area:

```
Sub Select_Range()
  Worksheets("Sheet1").Range(Cells(1, 2), Cells(1, 4)).Select 'selected B1:D1
End Sub
```

Or with variables:

```
Sub Combinate_Range_Cells()
Dim intRows, intColumn As Integer
  Worksheets("Sheet1").Activate
  intRows=InputBox("Count of rows")
  intColumns=InputBox("Count of column")
  Range(Cells(3, 3), Cells(2+intRows, 2+intColumn)).Activate
End Sub
```

1.4.2 FIND THE LAST CELL

The last described cell in the table has the following properties:

- **Rows.Count** last described cell in the column
- **Columns.Count** Last cell described in the line

	A	B	C
1	**Fruit**	**City**	
2	Apple	Stuttgart	
3	Melon	Paris	
4	Pear	London	
5			

Figure 10: Last column

```
Sub find_the_last_cell_row() 'counted from top to bottom
Dim strLastRow As String
  strLastRow = Cells(Rows.Count, 1).End(xlUp).Row
End Sub
```

⇨ The result is 4. "A4" is last described!

```
Sub find_the_last_cell_column () 'counted from left to right
Dim strLastColumn As String
  strLastColumn = Cells(1, Columns.Count).End(xlToLeft).Column
End Sub
```

⇨ The result is 2. "B1" is described last! The result is not a letter, but a number!

Download free eBooks at bookboon.com

1.4.3 INSERT A VALUE IN WRITABLE LAST CELL OF ROW

This often happens in practice. The lowest writable cell is entered as follows:

```
Sub insert_a_value_in_writable_last_cell_2()
Dim strLastRow As String
  strLastRow = Cells(Rows.Count, 3).End(xlUp).Row
  Range("C" & strLastRow).Offset(1, 0) = "I am the last value."
End Sub
```

	A	B	C	D
1	**Fruit**	**City**		
2	Apple	Stuttgart	I am the last value.	
3	Melon	Paris		
4	Pear	London		
5				

Figure 11: Last row

1.4.4 DETERMINE OF ACTIVE CELL, COLUMN, ROW AND ADDRESS

Now, let's find the active row, active column, and the active position of the cursor.

ActiveCell.Row gives us the line as a string
ActiveCell.Column gives us the column as a string
ActiveCell.Address gives us the address in range format with $ -sign before.

```
Sub Activecell_Infos()
    intRow = ActiveCell.Row
    intColumn = ActiveCell.Column
    Address = ActiveCell.Address
    Address_2 = ActiveCell.Address(RowAbsolute:=False, ColumnAbsolute:=False)
    MsgBox "The Cursor is in cell " & vbLf & _
    "Row: " & intRow & vbLf & _
    "Column: " & intColumn & vbLf & _
    "Address_1 " & Address & vbLf & _
    "Address_2 " & Address_2
End Sub
```

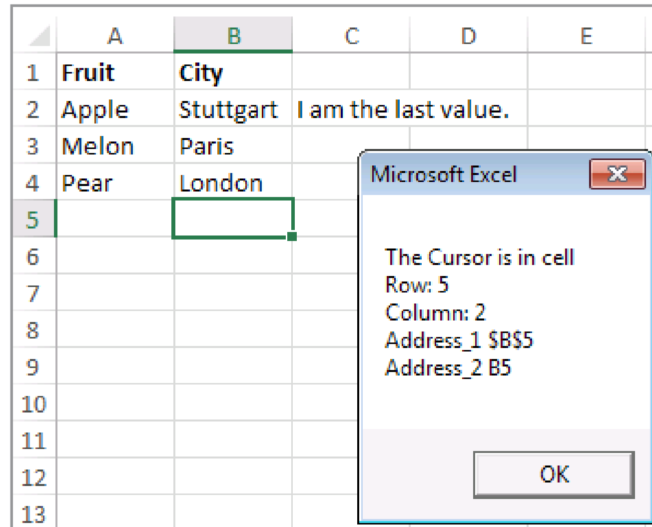


Figure 12: Cell information

To get the result without a $ sign, we set the attributes
RowAbsolte:=False or
ColumnAbsolute:=False.

In our example above looked like this:

⇨ Address_2 = ActiveCell.Address(RowAbsolute:=False, ColumnAbsolute:=False)

Enough with cells. Now we will look at the tables. ☺

1.4.5 DETERMINING AN ACTIVE SHEET

The following properties with the "**.Name**" statement determine the names of the active worksheet:

- **ActiveSheet.Name**
- **ActiveCell.Parent.Name**

```
Sub Determine_active_sheet_name()
  Dim strSheetName_1, strSheetName_2 As String
  strSheetName_1 = ActiveSheet.Name 'Variant 1
  strSheetName_2 = ActiveCell.Parent.Name 'Variant 2
  MsgBox strSheetName_1 & vbLf & strSheetName_2
End Sub
```

1.4.6 DETERMINING AN ACTIVE FILE

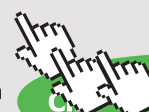
Now we come to the determination of the active file. To determine the name of the active file, we use the following property with the "**.Name**" statement:

- **ActiveWorkbook.Name**
- **ActiveSheet.Parent.Name**

```
Sub Determine_active_filename()
  Dim strFileName_1, strFileName_2 As String
  strFileName_1 = ActiveWorkbook.Name 'Variant 1
  strFileName_2 = ActiveSheet.Parent.Name 'Variant 2
  MsgBox strFileName_1 & vbLf & strFileName_2
End Sub
```

1.4.7 DETERMINING AN ACTIVE WINDOW

If we have multiple Excel files open at the same time, these files are then virtually or "digitally" stacked. We can use the "**ActiveWindow**" property to determine which window or Excel file is at the top. We can accomplish this with the "**ActiveWindow**" property appended to the "**.Caption**" property.

Deloitte.
Discover the truth at www.deloitte.ca/careers
© Deloitte & Touche LLP and affiliated entities.

Click on the ad to read more

```
Sub Determine_actives_window()
  Dim strWindow As String
  strWindow = ActiveWindow.Caption
  MsgBox strWindow
End Sub
```

Here we used some commands in a bunch. These have functions as follows:

Statement	Activity
ActiveCell.Address	Determine the address from active cell
ActiveCell.Row	Determine the row number from active cell
ActiveCell.Column	Determine the column number from active cell
ActiveCell.Parent.Name	Determine the name from active sheet of cursor position
ActiveSheet.Parent.Name	Determine the name from sheets
ActiveSheet.Name	Determine the name from active sheet
ActiveWindow	Determine the active window
ActiveWorkbook.Name	Determine the name from active workbook

Table 4: Same important Active-Property

1.4.8 SEARCH AND REPLACE OF CELL VALUE

Searching for and replacing specific content in the table is done using the following methods:

- **.Find**: Search in the area by specific term
- **.Replace**: Search in the area and replace with another term

First, we look for the **Range.Find**" "Search"!

We select certain areas for a value, here "Harun" is searched and the location is given.

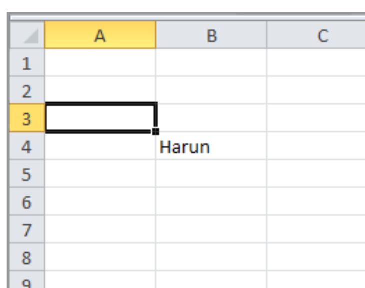


Figure 13: Search term

Download free eBooks at bookboon.com

The listing for them:

```
Sub Search_and_output_address()
Dim strAddress As String
  strAddress = Range("A1:C5").Find("Harun").Address
MsgBox strAddress
End Sub
```

And the result is:

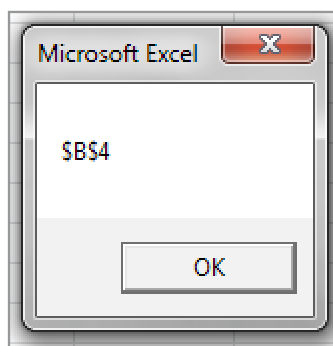


Figure 14: Result of search term

Now we search for the same term in several areas and find the location(s):

	A	B	C
1			
2			
3	Auto		
4		Harun	
5			Auto
6			
7		Auto	
8			
9			

Figure 15: Search term

The listing to:

```
Sub Search_and_output_address _2()
Dim rgRange As Range
Dim strAddress, strRange As String
Set rgRange = Cells.Find("Auto")
If Not rgRange Is Nothing Then
   strAddress = rgRange.Address
   Do
      Set rgRange = Cells.FindNext(rgRange)
        strRange = rgRange.Address & vbLf & strRange
   Loop Until (rgRange.Address = strAddress)
   MsgBox strRange
End If
End Sub
```

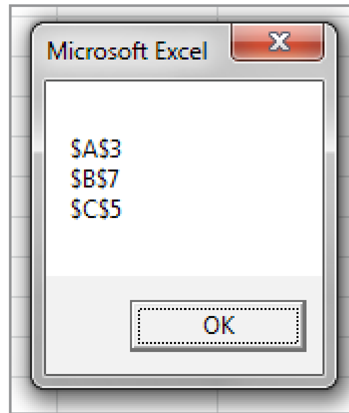


Figure 16: Result of search term

Now let's search for the values. When the values are found the cells will be highlighted or colored blue. Our table looks like this:

	A	B	C
1			
2	19		
3	Auto	35	
4		Harun	
5	21	3	Auto
6			
7		Auto	
8		20	
9			

Figure 17: Search number

All the cells in the specified range are searched for. In this example the range includes all numerical values larger than 20. When the respective cell is found it is filled in blue. The easiest way to find cell contents that can occur in multiple cells is most easily done through the For Each loop. For this we use the ">" greater / "<" less or "=" equals sign operator.

```
Sub Search_and_fill_with_colour()
Dim rngCell As Range
  For Each rngCell In Range("A1:C10")
     If IsNumeric(rngCell.Value) Then
      If rngCell.Value > 20 Then
        rngCell.Interior.ColorIndex = 5
      End If
     End If
  Next
End Sub
```

	A	B	C
1			
2	19		
3	Auto	35	
4		Harun	
5	21	3	Auto
6			
7		Auto	
8		20	
9			

Figure 18: Cell filled with color

1.4.9 REPLACING CELL VALUE

Now we will show a few examples demonstrating replacement. The following example is used to replace our cells containing "aut" with "Mercedes Benz". Again, we use the For Each loop. In addition, we will use the "Like" operator to compare content.

To search for cell contents we use the "Value" property. The contents of the cells can be different. They can be numeric, alphanumeric or formulas.

```
Sub Search_and_replace_1()
Dim rngCell As Range
  For Each rngcell In Range("A1:C10")
    If LCase(rngCell.Value) Like "aut*" Then
       rngCell.Value = "Mercedes Benz"
    End If
  Next
End Sub
```

Or with **Range.Replace**-Method, too:

```
Sub Search_and_replace_1 ()
Worksheets("Sheet").Range("A1:C10").Replace _
    What:="*aut*", Replacement:="Mercedes Benz", _
    SearchOrder:=xlByColumns, MatchCase:=True
End Sub
```

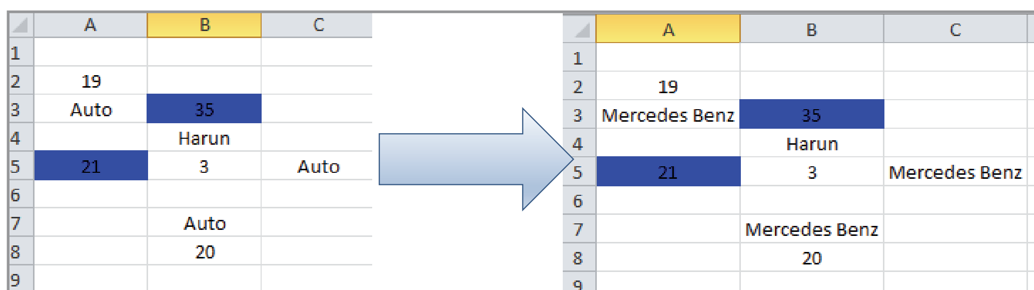


Figure 19: Result Search_and_replace_1

Each cell containing * ben * should be filled with the double frame, with the color "27". Again, we use the For Each loop. In addition, we use the "Like" operator to compare content.

```
Sub Search_and_replace_2()
Dim rngCell As Range
  For Each rngCell In Range("A1:C10")
   If LCase(rngCell.Value) Like "*ben*" Then
      rngCell.Borders.LineStyle = xlDouble
   End If
  Next
End Sub
```

	A	B	C
1			
2	19		
3	Mercedes Benz	35	
4		Harun	
5	21	3	Mercedes Benz
6			
7		Mercedes Benz	
8		20	
9			

Figure 20: Result Search_and_replace_2

All blue cells will change to yellow and be replaced with a dotted frame and the value "Daimler".

```
Sub Search_and_replace_3()
Dim rngCell As Range
  For Each rngCell In Range("A1:C10")
   If rngCell.Interior.ColorIndex = 5 Then
     With rngCell
          .Borders.LineStyle = xlDot
          .Interior.ColorIndex = 27
          .Value = "Daimler"
     End With
   End If
 Next
End Sub
```

	A	B	C
1			
2	19		
3	Mercedes Benz	Daimler	
4		Harun	
5	Daimler	3	Mercedes Benz
6			
7		Mercedes Benz	
8		20	
9			

Figure 21: Result Search_and_replace_3

Download free eBooks at bookboon.com

1.4.10 CLEAR OR DELETE OF INFORMATION IN SHEET

Cells can contain texts, numbers, formulas or even comments. We can format them with a border style, with a certain fill color or with a pattern. Everything entered into a cell can be deleted one by one or all at once with their formatting.

The following methods are responsible for this:

Delete statement	Description
Clear	Delete all entries and setting from a range
ClearContents	Delete only contents and formulas in an area
ClearComments	Delete only comments
ClearFormats	Delete only formats
Delete	Delete all cells

Table 5: Delete methods

```
Sub Delete_cell_formats()
        Worksheets("Sheet1").Range("A1").Clear
        Worksheets("Sheet1").Range("A2:C2").ClearContents
        Worksheets("Sheet1").Range("A3").ClearFormats
        Worksheets("Sheet1").Range(Cells(4, 1), Cells(4, 2)).ClearComments
        Worksheets("Sheet1").Range("A5").Delete
        ActiveCell.EntireRow.Clear 'Delete row
        ActiveCell.EntireColumn.ClearContents 'Delete column
End Sub
```

If you want to delete entire entries of the sheet, then as follows:

```
Sub Delete_all_Entries_of_sheet()
  Cells.ClearContents 'ClearContents deletes the contents and formulas one area
End Sub
```

Delete everything as follows:

```
Sub Delete_all_Entries_of_sheet()
  Cells.Delete
End Sub
```

Download free eBooks at bookboon.com

1.4.11 FILLING OF CELLS

Filling the cells or areas is usually done using VBA with the .Value property.

- These entries are in the respective cell of the active sheet:
  Range("A20").Value = "12345"
  Range("A20") = "12345"
  ActiveCell="Harun Kaplan"
  ActiveCell.Value="Harun Kaplan"

- This entry is explicitly entered in cell (A21) of sheet "VBA":
  Worksheets("VBA").Cells(21, 1).Value = "12345"

- This entry is entered in the renamed cell of a sheet. We will see similar examples a little later:
  Range("Renamed") = "Auto"
  Range("Renamed").Value = "Auto"

- Sum of the cells of the different sheets:
  WorksheetFunction.Sum(Worksheets("Sheet1").Range("A12:A14"), _
  Worksheets("Sheet2").Range("A15"))

```
Sub Fill_Cell()
Dim strName, strFruit As String
Dim intValue1, intSum As Integer

'Write Assignment in Cells
ActiveCell = "Harun Kaplan"
Range("A20") = "12345"
Worksheets("VBA").Cells(21, 1).Value = "12345"
Range("Renamed") = "Auto" '"B13" in Sheet "Sheet1"

'Define Cell contents as Variables
strName = Range("A1")
intValue1 = Worksheets("VBA").Cells(21, 1).Value = "12345"
strFruit = Cells(2, 1).Value
intSum = WorksheetFunction.Sum(Worksheets("Sheet1").Range("A12:A14"), _
           Worksheets("Sheet2").Range("A15"))

MsgBox ("Name : " & strName & Chr(10) & _
          "Value : " & intValue1 & Chr(10) & _
          "Fruit : " & strFruit & Chr(10) & _
          "Sum : " & intSum)
End Sub
```

In the next example, cells A10 to A1 are backfilled from 10 to 1.

```
Sub Fill_the_Value_with_loop()
Dim i As Integer
  For i = 10 To 1 Step -1
    Range(Cells(1, i), Cells(1, i)).Value = i
  Next i
End Sub
```

1.4.12 INSERT A FUNCTION OR A FORMULA

First, we take a result from a calculation (show here as the sum of two values) and place it in a particular cell. In the first example, each value is assi gned to a variable. In the second example it is shown directly without variables.

```
Sub Calculate_VBA_Code_Variable()
   Dim intValue1, intValue2, intResult As Integer
   intValue1= Range("B1").Value 'First value as variable
   intValue2= Range("C1").Value 'second value as variable
   intResult = intValue1 + intValue2 'The result of both
   Range("A1").Value = intResult 'Result in Cell "A1"
End Sub
```

Here without Variable:

```
Sub Calculate_VBA_Code_direct()
   Range("A1").Value = Range("B1").Value + Range("C1").Value
End Sub
```

However, it is possible that we will have to write a formula directly in the cell. We can enter this with the Formula property in the A1 reference system for the table macro language.

In the example below, a formula is inserted in cell "A1" of "Sheet1". Which in turn sums up the cells "B1" and "C1.

```
Sub Insert_Formula_1()
   Worksheets("Sheet1").Range("A1").Formula = "=$B$1+$C$1"
End Sub
```

A1 fx =$B$1+$C$1

	A	B	C
1	36	12	24

Figure 22: Filled formula

Next example adds the formula "= A1 * 0.5" to cell "B2" on all sheets

```
Sub Insert_Formula_2()
Dim intCount As Integer
 For intCount = 1 To Worksheets.Count
   Worksheets(intCount ).Cells(2, 2).Formula = "=A1*0.5"
 Next
End Sub
```

The following example adds an Excel function to the "Sheet1" of the current file in the cell "B16" of the formula „**=VLOOKUP(A16;'[Examplefile.xlsx]Sheet1**".

```
Sub Insert_Formula_3()
      ThisWorkbook.Worksheets("Sheet1").Range("b16").FormulaLocal = _
      "=VLOOKUP(A16;'[Examplefile.xlsx]Sheet1'"
End Sub
```

1.4.13 COPY, MOVE AND INSERT OF CELL OR RANGE

The VBA syntax for copying and cutting cells or ranges is written almost the same way. Cut will use "**.Cut**" and copy will use "**.Copy**". Everything else remains the same.

The areas A1: B3 are filled in the table "Data.

	A	B	C	D	E	F	G
1	Fruit	City		Fruit	City		
2	Apple	Stuttgart		Apple	Stuttgart		
3	Melon	Paris		Melon	Paris		
4	Pear	London					
5							
6							

VBA | Orange | Apple | Melon | Pear | Books

Figure 23: Example Copy

This example should copy the range "A1: B3" and paste it from "D1":

```
Sub Range_Copy_Paste_1()
  Range("A1:B3").Copy Range("D1")
End Sub
```

Here is the same example with "**.Cut**". The area will be cut out and inserted starting from "D1":

```
Sub Range_Cut_Paste_1()
  Range("A1:B3").Cut Range("D1")
End Sub
```

We add the same area to the "Apple" table from cell A1:

```
Sub Range_Copy_Paste_2()
  Range("A1:B3").Copy Worksheets("Apple").Range("A1")
End Sub
```

The area ("A1: B3") from the "Orange" sheet is inserted in the "Melon" sheet starting with cell A1:

```
Sub Range_Copy_Paste_3()
  Worksheets("Orange").Range("A1:B3").Copy _
  Worksheets("Melon").Range("A1")
End Sub
```

We add the area ("A1: B3") from the "Data" table of the "Data.xlxs" file to the "Evaluation" table of the "Evaluation.xlsx" file from cell A1:

```
Sub Range_Copy_Paste_4()
  Workbooks("Data.xlsx").Worksheets("Data").Range("A1:B3").Copy _
  Workbooks("Evaluation.xlsx").Worksheets("Evaluation").Range("A1")
End Sub
```

1.4.14 SELECT A RANGE WITH "STRG+SHIFT+*"

In Excel or VBA there are two ways to mark an area:

First way:
Excel: Position the cursor on the top edge, then drag it to the diagonally opposite edge.
VBA: Range ("A1: D4"). Select

Second way:
Excel: Position the cursor somewhere in the respective area, then highlight it with the key combination Ctrl + Shift + *. This elegant way can also be used via VBA as follows
VBA: **Selection.CurrentRegion.Copy**

```
Sub Range_Copy_Paste_5()
    Range("A1").Select
    Selection.CurrentRegion.Copy Range("D1")
End Sub
```

With this type of field, as shown below, it is sufficient to position the mouse somewhere in the field "A1: D4". All fields that touch the edge or the corners are highlighted.

Download free eBooks at bookboon.com

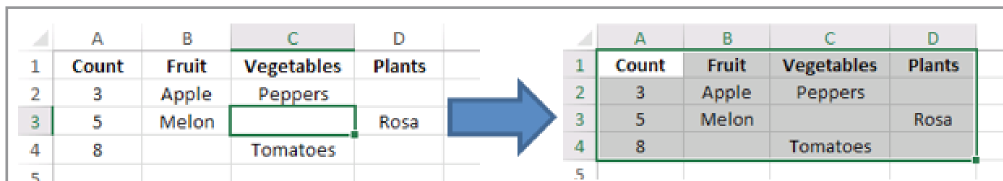


Figure 24: Select with „Strg+Shift+*" in VBA -Code

```
Sub Range_Copy_Paste_6()
    Range("C1").Select
    Selection.CurrentRegion.Copy Worksheets("Books").Range("A11")
End Sub
```

1.4.15 SELECT A RANGE WITH SET

The areas can also be referenced with the "Set" statement:

```
Sub Range_Copy_Paste_6()
  Dim rngSource As Range
  Dim rngTarget As Range

  Set rngSource = Workbooks("Data.xls").Worksheets("Data").Range("A1:B3").Copy
  Set rngTarget = Workbooks("Evaluation.xls").Worksheets("Evaluation").Range("A1")

        rngSource.Copy rngTarget

  Set rngSource = Nothing
  Set rngTarget = Nothing
End Sub
```

1.4.16 RENAMING A CELL OR A RANGE

I would like to show you a very useful function. Here are a few rules to keep in mind: It may ONLY be used once per file.

It can't be assigned to a macro or control.Otherwise, the cursor will jump directly into the existing cell or to the macro or to the control.

C4 | Example_3

	A	B	C	D
1	Position	Name	First name	Age
2	1	Schmidt	Example_1	20
3	2	Mayer	Example_2	21
4	3	Müller	Example_3	19

Figure 25: Cell names

In the upper figure, the cursor is in cell "C4".

In Excel, all columns, rows, and cells have their own names. The lines are labeled with the numbers between 1 - 1,048,576 and the columns are labeled with the letters from A - XFD. The names of the cells of a table are designated from A1 to XFD1048576.

Therefore, when renaming old cells new names are given. If we have given the cell a new name, then the new cell name appears. This new name can then be addressed later with Range ("new_name").

Script looks like this:

```
ActiveWorkbook.Names.Add Name:="New_Name" , RefersTo:="Cell or Range"

Sub Rename_a_cell_1()
ActiveWorkbook.Names.Add _
  Name:="Fruit", _
  RefersTo:=Worksheets("Books").Range("B1")
 Range("Fruit").Select
End Sub
```

Fruit | Fruit

	A	B	C	D	E	F
1	Count	Fruit	Vegetables	Plants		
2	3	Apple	Peppers			
3	5	Melon		Rosa		
4	8		Tomatoes			

VBA | Orange | Apple | Melon | Pear | Books

Figure 26: Renamed cell

Now for my first practical example:

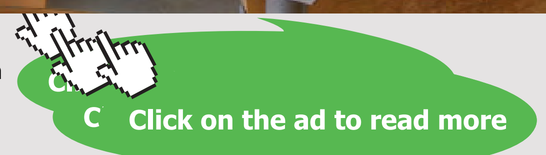
Often, a part of a source or a database must be read into a target file. This could look like the example shown below. The data from the active line from the file "source.xlsm" should be transferred to the "target.xlsm" file.

For our example, we leave the data from the "Name" columns; "First name" and "Age" are transferred and then further extended with the columns "Occupation".

In our example below, we see the previously described procedure transfer data in the correct order. It works until there are no newly inserted columns between "A" and "B" or between "B" and "C". Otherwise the data will be entered in empty columns.

In the file "target.xlsx" the cells are changed by hand as follows:
„A1" in „Target_Name";
„B1" in „Target_Firstname" and
„C1" in „Target_Age".

be > your degree
Bring your talent and passion to a global organization at the forefront of business, technology and innovation. Discover how great you can be.
Visit accenture.com/bookboon
Be greater than.
consulting | technology | outsourcing
accenture
High performance. Delivered.
© 2013 Accenture. All rights reserved.



If we insert new columns later then this information will be entered in the correct column.

How do we determine the right column?

The Range ("Target_name"). Column determines the column of the cell from "Target_name".

Is the Cell statement written? Cells (row, column).

We still need the right line:

Lastline = Cells (Rows.Count, 1) .End (xlUp) .Row + 1

We put them together like this:
Cells (Lastline, Range ("Target_Name"). Column) .Value

The transfer now works forever with the following VBA syntax.
Cells (Lastline, Range ("Target_name"). Column) .Value = Target_name

Here is our example:

First a screenshot of what they should look like:

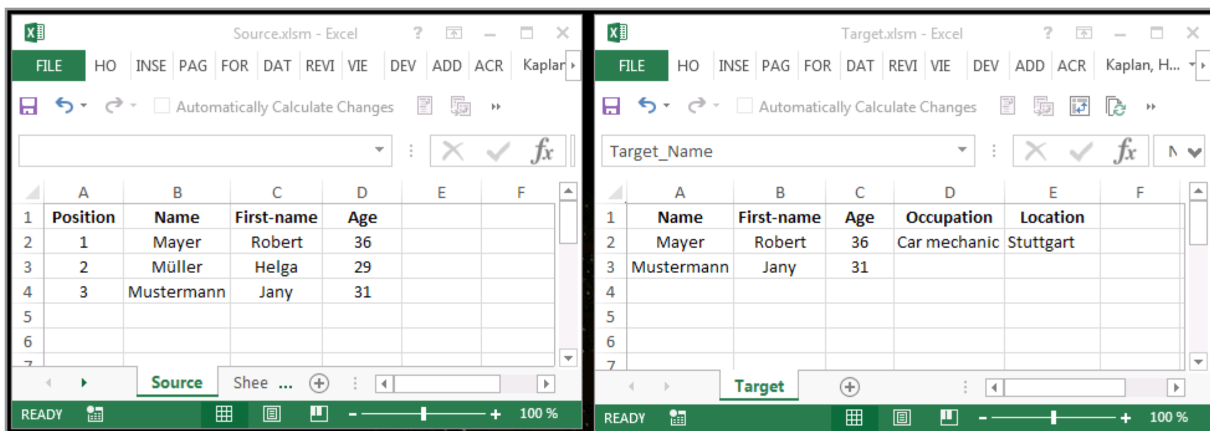


Figure 27: Source and Target file

This is what our example looks like:

```
Sub Transfer_Data()
Application.ScreenUpdating = False
On Error Resume Next
    Active_File = ActiveWorkbook.Name
    Target_Filename = "Target.xlsm"
    activecell_row = ActiveCell.Row
'Read of Data
    Target_Name = Cells(activecell_row, 2).Value
    Target_Firstname = Cells(activecell_row, 3).Value
    Target_Age = Cells(activecell_row, 4).Value
'Transfer of Data
    Workbooks(Target_Filename).Activate
    Worksheets("Target").Activate
    LastLine = Cells(Rows.Count, 1).End(xlUp).Row + 1
    Cells(lastLine, Range("Target_Name").Column).Value = Target_Name
    Cells(lastLine, Range("Target_Firstname").Column).Value = Target_Firstname
    Cells(lastLine, Range("Target_Age").Column).Value = Target_Age
    Workbooks(Active_File).Activate
End Sub
```

Now add a new column between these columns and start again. You will see that it works perfectly.

Another practical example:

This example shows us how an area is automatically renamed. Subsequently, this area is applied.

I will use this possibility in UserForm. In our example, a column, shown here as "J", is automatically filled with directory names each time the file is started. This means that if new phrases have been created or deleted, they will be loaded accordingly.

You can examine examples with directories and UserForm in detail in the next books.

The figure is shown as follows:

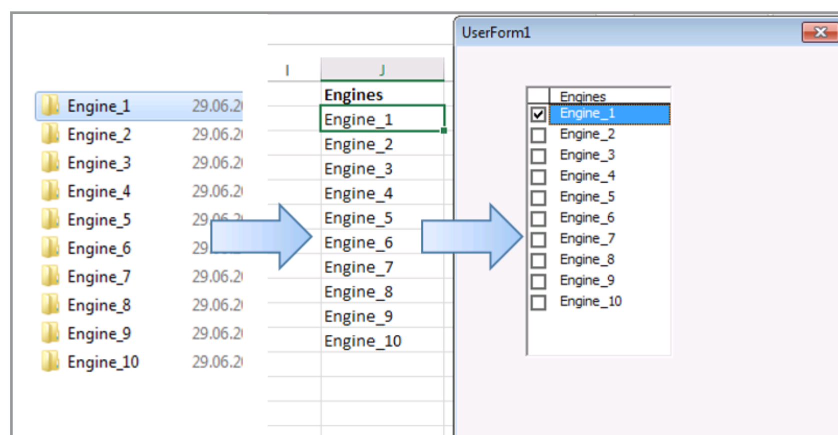


Figure 28: Directory in Explorer > Entries in Excel > Looks in UserForm

In this section, only the renaming and sorting of the area was listed

```
Sub Example_of_AddName()
   Range("J2").Select
'Dynamics Entries
   ActiveWorkbook.Names.Add _
     Name:="Engine_Number", _
     RefersTo:=Worksheets("Target").Range(Selection, Selection.End(xlDown))
   Range("J2").Select
'Count of Engine > selection range > Sort on
   Count_of_engine = Cells(Rows.Count, 10).End(xlUp).Row
   Range(Selection, Selection.End(xlDown)).Select
   ActiveWorkbook.Worksheets("Target").Sort.SortFields.Clear
   ActiveWorkbook.Worksheets("Target").Sort.SortFields.Add Key:=Range("J2"), _
       SortOn:=xlSortOnValues, Order:=xlAscending, DataOption:= _
       xlSortTextAsNumbers
   With ActiveWorkbook.Worksheets("Target").Sort
       .SetRange Range("J2:J" & Count_of_engine)
       .Header = xlNo
       .MatchCase = False
       .Orientation = xlTopToBottom
       .SortMethod = xlPinYin
       .Apply
   End With
End Sub
```

1.4.17 DELETE A RENAMED CELL

We can also delete the new name or undo renaming and the cell will be shown with its original name.

ActiveWorkbook.Names("renamed_cell")**.Delete**

We had previously renamed the cell "A1" to "fruit". The unnamed cell is deleted with the "**.Delete**". After the deletion, the cell is called its original name again "A1".

```
Sub Delete_a_renamed_cell()
  ActiveWorkbook.Names("Fruit").Delete
End Sub
```

1.4.18 SELECT A RENAMED RANGE

In our example we see column "A" filled with some entries. We will mark this area using VBA code and rename it "Vehicles". Then we will the mark new area with the Select statement.

Here is the listing and the appearance of the result:

```
Sub Rename_a_range()
  Dim rngRange As Range
  Dim rngEnd As Range
  Sheets("Sheet1").Activate
  Set rngRange = Range(Cells(2, 1), _
                     Cells(Cells(Rows.Count, 1).End(xlUp).Row, 1))

  ActiveWorkbook.Names.Add _
        Name:="Vehicle", _
        RefersTo:=rngRange
  rngRange.Select
End Sub
```

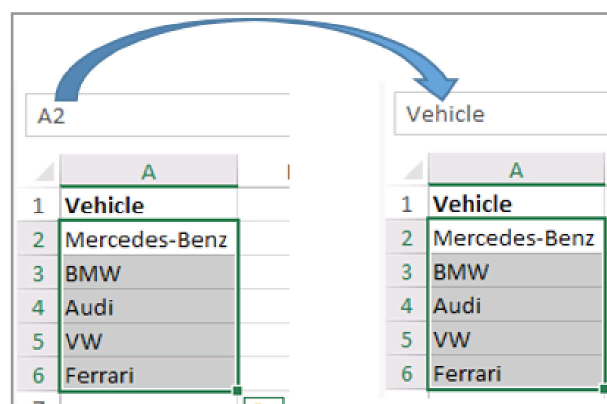


Figure 29: Renamed Range 1

1.4.19 INSERT A CELL, A COLUMN OR A ROW

The insertion of cells, rows and columns is done with the following properties:

- Insert **.EntireRow** lines
- Insert **.EntireColumn** columns.

With the argument **Shift: =**" we indicate in which direction the active cells or the remaining cells should be moved. The direction is controlled by the integrated constants "**xlDown**" or "**xlToRight**".

```
Sub Insert()
 Range("B2").Activate
 ActiveCell.EntireRow.Insert          'insert a row
 ActiveCell.EntireColumn.Insert       'insert a column
 ActiveCell.Insert Shift:=xlToRight  'insert a row and. Remaining cells are moved to the right
End Sub
```

Download free eBooks at bookboon.com

1.4.20 HIDDEN A COLUMN OR A ROW

Fade in or hiding a column or row visible is done using the following keywords:

- True for hiding
- False appended to the .Hidden property for a fade in.

```
Sub Hidden_true_false()
  With Rows("1:2") 'for Rows
         .EntireRow.Hidden=True
         .EntireRow.Hidden=False
  End With
'
  With Columns("A:A") 'for Column
         .EntireColumn.Hidden=True
         .EntireColumn.Hidden=False
  End With
End Sub
```

1.5 HANDLING "SPECIALCELLS"

Let's look at some "special" methods, also referred to as "SpecialCells". These methods greatly facilitate our lives. Our examples will explain more than our words.

Once we've entered a round-brace using the SpecialCells method, an Intelligent window pops up and an xl key can be selected accordingly. We will look at a few important examples for us, which are also examples that can be used in practice.

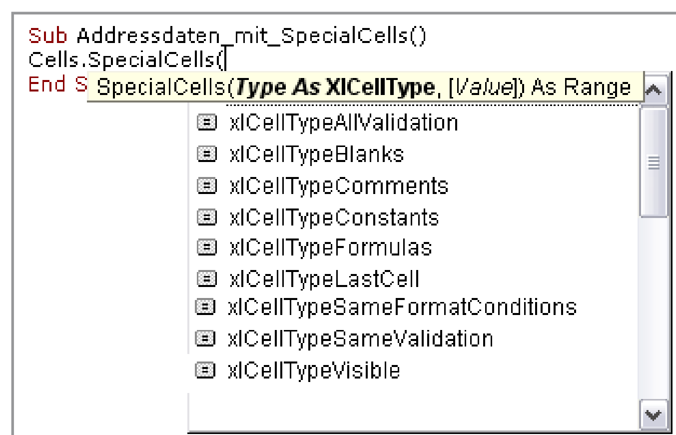


Figure 30: xl-Keys from SpecialCells

Our example:

Our table includes

- Comments.
- Formulas.
- Active cell is (J20)

Then let's use our example.

The active address is shown as a fixed reference, i.e. with the dollar sign: $J$20
The line is displayed as a value: 20
The column is represented as a value: 10 => "J" is the 10th letter.
The cells with comments are shown as solid references: $J$6; $E$9, $I$10, $H$12.
The cells with formulas are also shown as a fixed reference: $G$10;$F$16.

```
Sub Cell_Addresses_with_SpecialCells()
  Debug.Print Cells.SpecialCells(xlLastCell).Address
  Debug.Print Cells.SpecialCells(xlLastCell).Row
  Debug.Print Cells.SpecialCells(xlLastCell).Column
  Debug.Print Cells.SpecialCells(xlCellTypeComments).Address
  Debug.Print Cells.SpecialCells(xlCellTypeFormulas).Address
End Sub
```

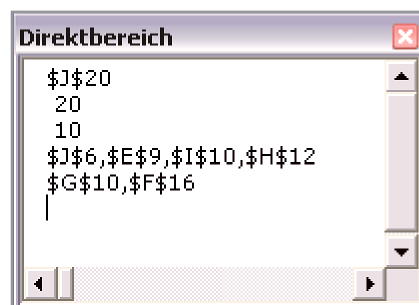


Figure 31: Cell information with SpecialCells

1.5.1 FIND THE LAST CELL IN ROW OR IN COLUMN

We have already determined in the chapter that ActiveCell is the last cell of a certain row or a certain column. Now we can also find the entire table of the last cell in rows or in columns. For this we use "**SpecialCells (xlCellTypeLastCell)**".

The lowest cell of the described table will determine this:

```
Sub Lastcell_in_a_row()
Dim intRow As Integer
  intRow = ActiveSheet.Cells.SpecialCells(xlCellTypeLastCell).Row
End Sub
```

The right-hand column of a table will determine hereby:

```
Sub Lastcell_in_a_column()
Dim intColumn As Integer
 intColumn = ActiveSheet.Cells.SpecialCells(xlCellTypeLastCell).Column
End Sub
```

1.5.2 CLEARING ALL COMMENTS IN A SHEET

We will go into more detail in the next section. For all of the SpecialCells functions to be together, we will look at two examples with "**.ClearComments**". Both examples show us how we can delete all the comments in a worksheet in one go.

```
Sub Delete_all_Comments_in_a_sheet()
  With ActiveSheet
    If .Comments.Count > 0 Then
      .Cells.SpecialCells(xlCellTypeComments).ClearComments
    End If
  End With
End Sub
```

Second example:

```
Sub Delete_all_comments_in_a_Workbook()
Dim wsSheet As Worksheet
  For Each wsSheet In Worksheets
    With wsSheet
      If .Comments.Count > 0 Then
        .Cells.SpecialCells(xlCellTypeComments).ClearComments
      End If
    End With
  Next wsSheet
End Sub
```

1.6 HANDLING COMMENTS

In this chapter, we learn to deal with comments. This is a very useful tool in professional life. Information from many different cells can be summarized within a comment and made visible with a mouse movement.

1.6.1 INSERT A COMMENT

The comments function is very useful tool. This allows us to group a lot of information into one cell. The comment content is unfortunately not evaluable.

We can add a comment with important information to each cell with the AddComment function. As soon as we move the cursor over the cell, our comment becomes visible.

I have adapted my database for this book. Let's take a look. The file will be named "Comment_Source.xlsm". It contains important information for each vehicle. The information from the file "Comment_Source.xlsm" should be visible as a comment in the file "Comment_Target.xlsx".

We have no control mechanisms here, such as whether the cell has a comment, or the target file is open, or whether information to be transmitted has already been transferred.

Here we open both files. We position our cursor in the line that should be transferred. Then click on the Smiley button. The data is now transferred.

In the figure, line 4 has been transferred.

Shown here is our source file, as well as the target file with a comment.

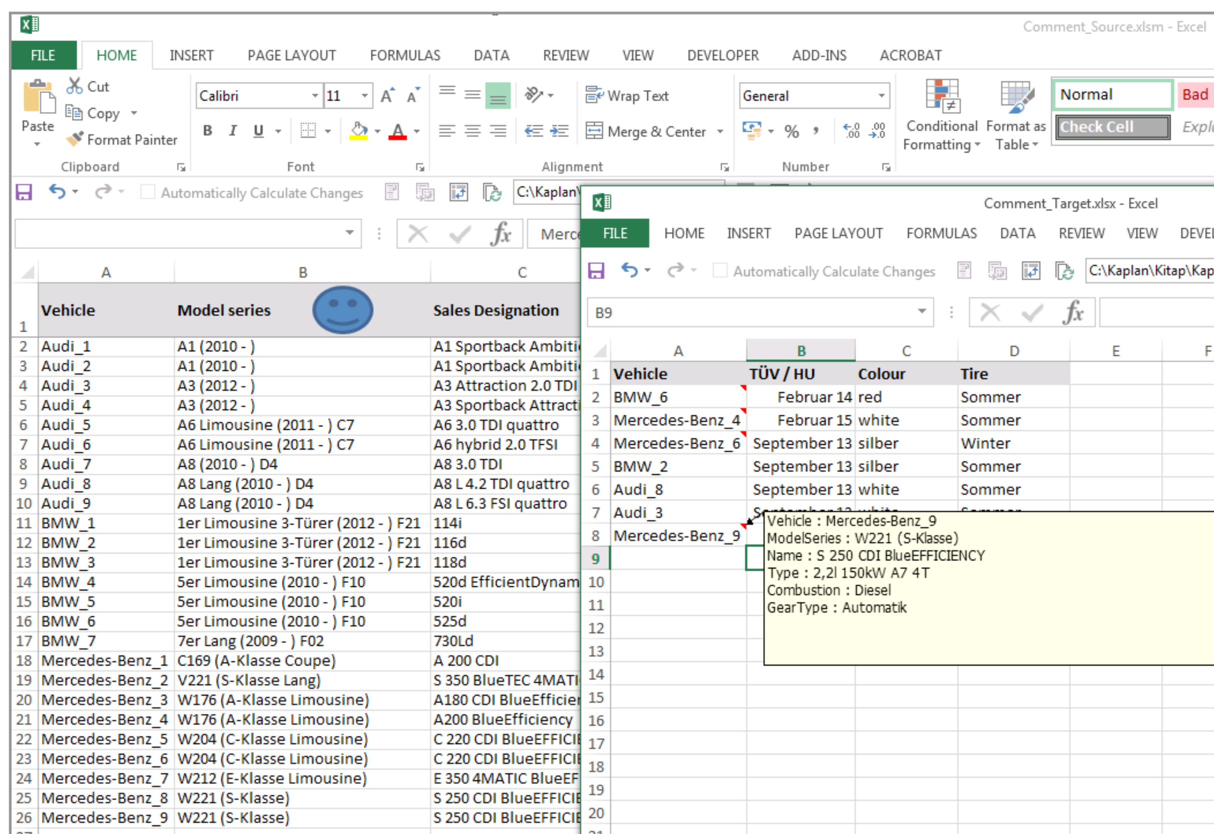


Figure 32: Example comments with more information

And the listing for then:

```
Sub Vehicle_Data_Transfer()

'Declar Variable
Dim strVehicle, strModelSeries, strName, strType, strCombustion, strGearType As String
Dim intActive_Line, intLast_Line As Integer

'Activity in the Source-Data
'Select data to be transferred
intActive_Line = ActiveCell.Row

'Read Motor Data in the current line
strVehicle = Cells(intActive_Line, 1).Value
strModelSeries = Cells(intActive_Line, 2).Value
strName = Cells(intActive_Line, 3).Value
strType = Cells(intActive_Line, 4).Value
strCombustion = Cells(intActive_Line, 5).Value
strGearType = Cells(intActive_Line, 6).Value

'Activity in the Target-Data
'Target-Data activation
Workbooks("Comment_Target.xlsx").Activate

'Identify Last writable Line
intLast_Line = Cells(Rows.Count, 1).End(xlUp).Row + 1
'and Select
Cells(intLast_Line, 1).Select

'Activate Last writable Line and
  With ActiveCell
    .Value = strVehicle 'Enter Vehicle
    .ColumnWidth = 15 'Set Column Width at 15
      With .AddComment 'Set Height and Width of the comment section
        .Shape.Height = 100
        .Shape.Width = 300
      End With
          'Fill Comment Section
    .Comment.Text Text:="Vehicle : " & strVehicle & vbNewLine & _
                                   "ModelSeries : " & strModelSeries & vbNewLine & _
                                   "Name : " & strName & vbNewLine & _
                                   "Type : " & strType & vbNewLine & _
                                   "Combustion : " & strCombustion & vbNewLine & _
                                   "GearType : " & strGearType
    ActiveCell.Offset(1, 0).Select
  End With
End Sub
```

1.6.2 CLEARING OF COMMENTS

The comments can be deleted in two ways:

- Comment.Delete
- Clear Comment

We should prefer the **ClearComment** statement. For the execution of the command will be executed regardless of whether that cell has a comment or not.

The **Comment.Delete** statement, on the other hand, returns an error message if the cell does not contain a comment.

```
Sun Delete_a_comment()
  If Range(„B2").Comment Is Nothing Then
    MsgBox "Here in this cell is no commend."
  Else
    Range („B2").ClearComments
  End If
End Sub
```

To delete all comments at once, proceed as follows:

```
Sub Delete_all_commentars()
  With ActiveSheet
    If .Comments.Count > 0 Then
      .Cells.SpecialCells(xlCellTypeComments).ClearComments
    End If
  End With
End Sub
```

1.7 SETTING OF CELL

A cell has extensive formatting options. When we right-click in a cell and select "Format Cells," a "Format Cells" window will appear with six different tabs. Of course, it will also be visible above the "Home" menu.

Download free eBooks at bookboon.com

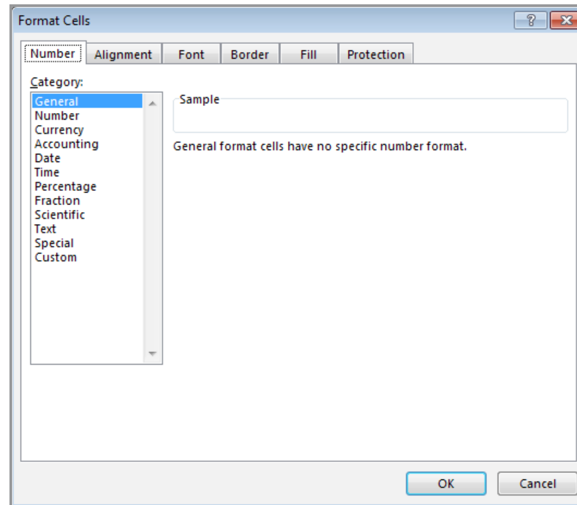


Figure 33: Window of Format Cells

A cell / area can have the following contents:

- Texts
- Numbers
- Formulas
- Comment

This content can be formatted according to the following criteria:

- Alignment
- Font
- Frame
- Color
- Protection

Now we look at the listed criteria with examples.

1.7.1 SETTING OF FONTS

The fonts are formatted with the "**.Font**" property. The syntax starts with a formatting area, followed by the font command, then its property, and finally the value.

Property	Description
.Bold	Bold for True
.Color	Font colour mixed RYB (Red/Yellow/Blue)
.ColorIndex	Font colour Index 1-56
.FontStyle	Font style
.Italic	Cursive for True
.Name	Font Name
.SubScript	Subscript for True
.SuperScript	Superscript for True
.UnderLine	Underline for True

Table 6: Text/Font-Settings

We use the With-structure here to combine several properties at the same time.

Our first example:

The fonts for all cells in Table1 should be:

- Arial
- 10 point size
- shown in green font

I'll show the example first without the With-structure. This type of programming may be confusing.

```
Sub Setting_Font_without_With()
  Worksheets("Sheet1").Cells.Font.Size = 10
  Worksheets("Sheet1").Cells.Font.Name = "Arial"
  Worksheets("Sheet1").Cells.Font.ColorIndex = 4
End Sub
```

It is even clearer in a With-structure:

```
Sub Setting_font_1()
  With Worksheets("Sheet1").Cells.Font
        .Size = 10
        .Name = "Arial"
        .ColorIndex = 4
  End With
End Sub
```

Here is an example within an area using "A7:A11".

```
Sub Setting_font_2()
Dim last_cell As Integer
Last_cell = Cells(Rows.Count, 1).End(xlUp).Row
With Worksheets("Sheet1").Range(Cells(7, 1), Cells(last_cell, 1)).Font
  .Size = 14
  .Bold = True
  .Name = "Arial"
  .ColorIndex = 4
End With
End Sub
```

If we only want to assign one property, we can enter it one line at a time. For example, we would like to set the font sizes of cells A1 through C5 in Table 1 to 12 points:

```
Sub Setting_font_3()
   Worksheets("Sheet1").Range(Cells(1, 1), Cells(5,3)).Font.Size = True
End Sub
```

In our next example we would like to check a formatting, font size> 5. If necessary, assign new properties, font size = 5.

```
Sub Font_size_check()
Dim rngFont As Range
  For Each rngFont In Range("A1:B5")
    If rngFont.Font.Size > 10 Then
         With rngFont.Font
            .Size = 8
            .Italic = False
         End With
    End If
  Next
End Sub
```

1.7.2 SETTING OF BORDERS

The frames of a cell / range are formatted using the "**.Borders**" property or the "**BorderAround**" method, respectively. The Borders property can also be explicitly selected with an index value for a particular page. Without an index value, all pages are formatted.

The BorderAround method formats all pages of a cell / range. The properties are indicated with a colon and an equal sign.

Property	Description
.Color	Colour from RGB (Red/Green /Blue)
.ColorIndex	Border colour with Index 1-56
.LineStyle xlContinuous xlDash xlDouble xlDot xlNone	Border type xl xl xl xl Non border
.Weight xlHairline xlMedium xlThick xlThin	Border weight Hairline medium thick thin

Table 7: Property of Borders

Download free eBooks at bookboon.com

In the following example, all pages of the area “A1: B7” are given a thin and blue colored frame.

```
Sub Border1()
  With Range("A1:B7").Borders
     .LineStyle = xlContinuous
     .ColorIndex = 5
     .Weight = xlThin
  End With
End Sub
```

However, if we only want to format a certain page of a cell / area, then we will select each page as an index in parentheses after the .Borders.

Index	Description
xlTop	Top side a cell
xlBottom	Bottom side a cell
xlLeft	Left side a cell
xlRight	Right cell a cell

Table 8: Index of a range

In the following example, the area B1: B5 only has lower sides with double and blue colored edges.

```
Sub Borders2()
 With Range("B1:B5").Borders(xlBottom)
   .LineStyle = xlDouble
   .ColorIndex = 5
 End With
End Sub
```

1.7.3 SETTING THE COLOR

We have already assigned colors to fonts and frames. The cells or areas themselves can also be displayed in color.

The following properties are used for this:

Download free eBooks at bookboon.com

Property	Description
.Color	Colour from RGB (Red / Green / Blue)
.ColorIndex	Colour from Index 1-56
.Pattern	Pattern type
.PatternColorIndex	Colour index for pattern

Table 9: Property of border filling

Our first example fills the area "A2: C7" in green.

```
Sub Fill_with_colour_1()
  Range("A2:C7").Interior.ColorIndex = 4 'green
End Sub
```

This example checks the color of the active cell, adding the value to the "intColor" variable. This is then checked in the if query. If true; Then the fill colors of the active and upper and lower cell are changed, as well as their frame.

```
Sub Fill_with_color_2()
Dim intColour As Integer
Range("B3").Activate
intColour = ActiveCell.Interior.ColorIndex
 If intColour = 4 Then
    With ActiveCell
      .Interior.ColorIndex = 17
      .Offset(-1, 0).Interior.ColorIndex = xlNone
      .Offset(1, 0).Interior.ColorIndex = xlNone
      .Borders.LineStyle = xlDouble
      .Borders.ColorIndex = 25
    End With
 End If
End Sub
```

Same example with RGB information.

Download free eBooks at bookboon.com

```
Sub Fill_with_colour_3()
Dim intColour As Integer
Range("B3").Activate
intColour = ActiveCell.Interior.ColorIndex
 If intColur = 17 Then
   With ActiveCell
     .Interior.Color = RGB(125, 200, 20)
     .Offset(-1, 0).Interior.ColorIndex = xlNone
     .Offset(1, 0).Interior.ColorIndex = xlNone
     .Borders.LineStyle = xlContinuous
    .Borders.Color = RGB(30, 160, 120)
    .Font.Color = RGB(130, 0, 150)
  End With
 End If
End Sub
```

The next example is not only filled with a color, but also uses a color pattern.

```
Sub Fill_with_color_4()
With Range("A2:C7")
   With .Interior
     .ColorIndex = 4 'green
     .Pattern = xlGray8
     .PatternColorIndex = 35
   End With
   .BorderAround ColorIndex:=28
End With
End Sub
```

1.8 OFFSET (NEW POSITION) OF CURSOR

I like working with the command “**Activecell.Offset**”. Activecell.Offset sets the cursor position to “zero”. Here’s the new theoretical “A1” of the cursor. I tried to show how it works below.

From the zero point, columns are indicated to the right without signs, to the left with minus signs while the lines to the bottom are indicated without signs, and the top with minus signs.

With the indication .Value we can read values from the cell

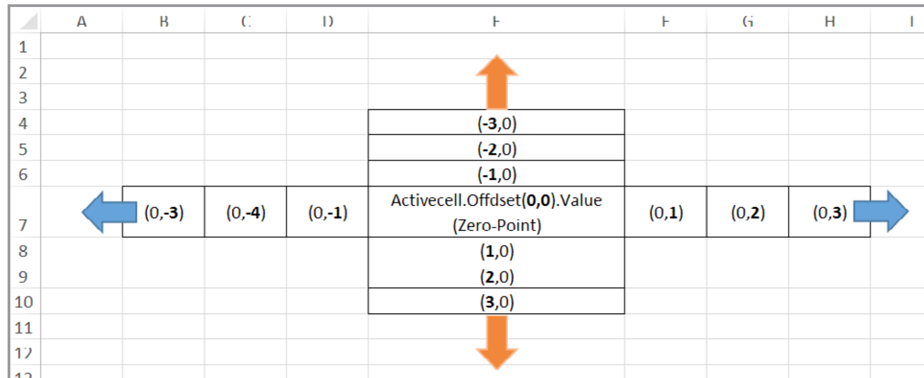


Figure 34: Functionality of Activecell.Offset

With Activecell.Offset, you can easily handle everything from simple to very complex, repetitive tasks. Now let's look at an example of how it works:

Our active cell in our example is the cell (C3). There the data "name; First name and age" are read and inserted somewher.

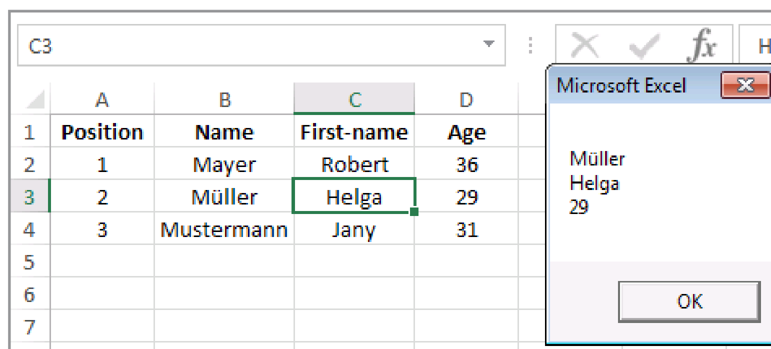


Figure 35: Example for .Offset

Before the data is inserted, our listing might look like this:

```
Sub Example_for_Activecell_Offset()
'Read of value
  Source_Name = ActiveCell.Offset(0, -1).Value
  Source_Firstname = ActiveCell.Offset(0, 0).Value
  Source_Age = ActiveCell.Offset(0, 1).Value
'From here the values are inserted somewhere
MsgBox Source_Name & vbLf & Source_Firstname & vbLf & Source_Age
End Sub
```

1.9 CREATED A LINK

Here we will see examples with simple and composite links. This link is created with the "**.Hyperlinks.Add**" function. When a hyperlink is created, it will be displayed in blue and underlined text.

```
Sub Unformatted_Link()
  With Worksheets("Sheet1")
   .Hyperlinks.Add .Range("A1"), "http://example.microsoft.com"
  End With
End Sub
```

The result is:

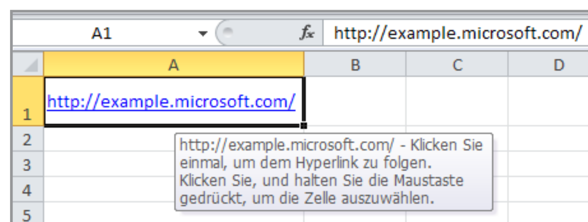
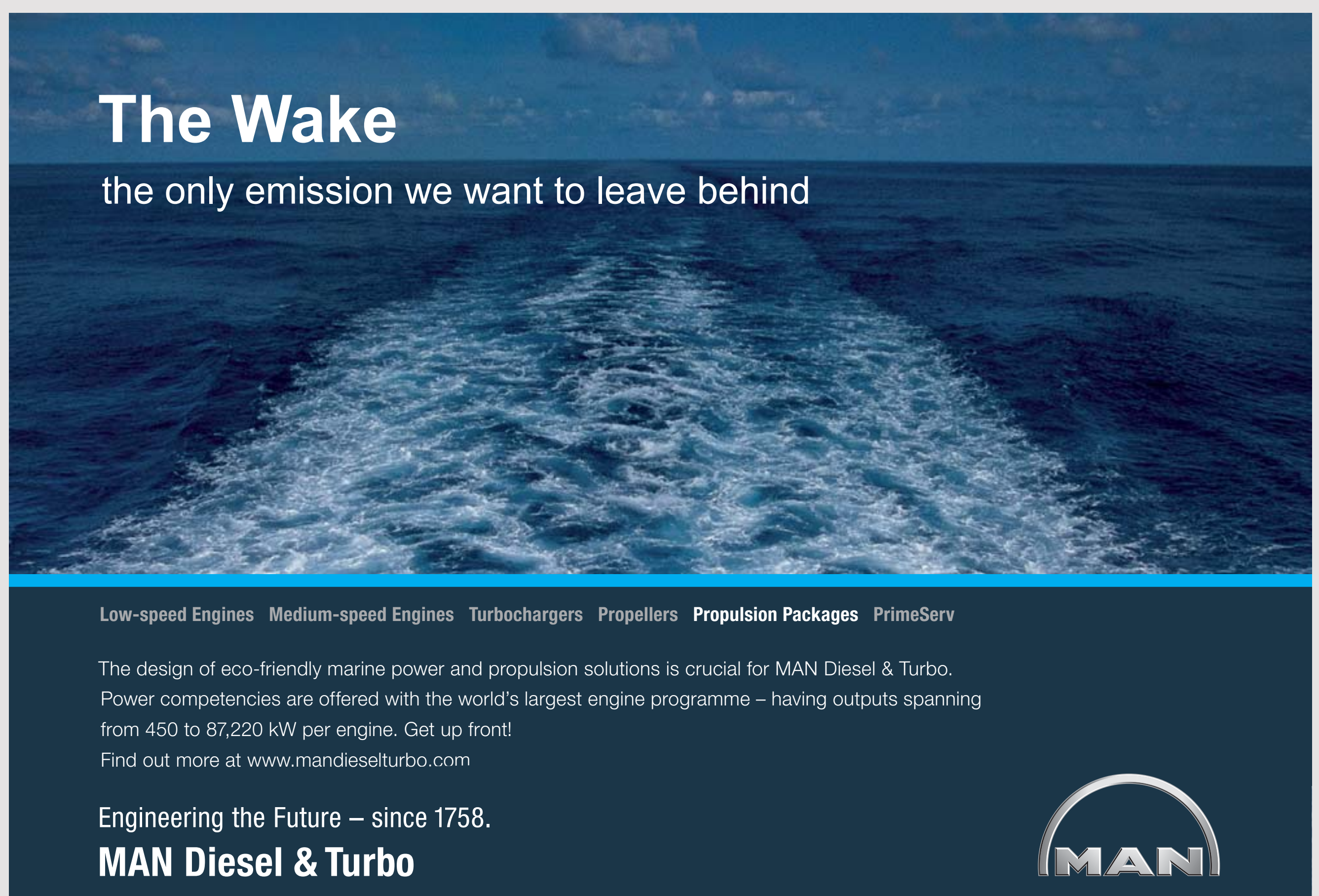


Figure 36: Unformatted Link



Download free eBooks at bookboon.com

But, if we want to format the link, we can do it like this:

```
Sub Formatted_link()
With Worksheets("Sheet1")
.Hyperlinks.Add .Range("A1"), "http://example.microsoft.com"
   With Range("A1").Font
      .Name = "Arial"
      .FontStyle = "Kursiv"
      .Size = 12
      .ColorIndex = 3
   End With
End With
End Sub
```

The result is:

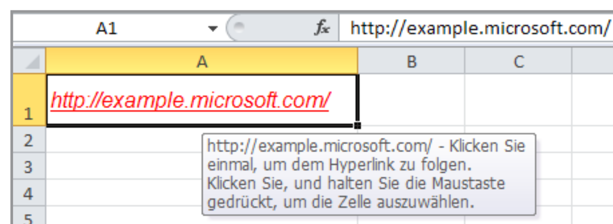


Figure 37: Formatted Link

1.10 SETTING THE BEST ROWS WIDE AND BEST COLUMN HEIGHT

By default, a row height is set to 15 and a column width is set to 10.71. The entered texts are usually longer or higher than the default values. If we have filled a lot of cells with different lengths of text, one or the other texts may not be completely readable. If these cells are automatically filled and then printed, then an automatic row height and column width is very useful.

To do this, we use the "**.AutoFit**" method. The column width or row height in the specified range is changed using the AutoFit method.

Here are a few examples:

The width of columns A to F in Table 1 is optimized with:
Worksheets("Sheet1").Columns("A:F").AutoFit

The width of the columns of the area "A6: F115" in Table1 are optimized with:
Worksheets("A6:F115").Columns("A6:F115").AutoFit

```
Sub ColumnWidthRowHeight_EntirePage_automatic_setting_1()
  With ActiveSheet.UsedRange 'Applied Area of active page
    .Columns.AutoFit
    .Rows.AutoFit
  End With
End Sub

Sub ColumnWidthRowHeight_EntirePage_automatic_setting_2()
ActiveCell.CurrentRegion.Select
  With Selection
    .Columns.AutoFit
    .Rows.AutoFit
  End With
End Sub

Sub ColumnWidthRowHeight_EntirePage_automatic_setting_3()
  With ActiveSheet.Range("A6:F115")
    .Columns.AutoFit
    .Rows.AutoFit
  End With
End Sub

Sub ColumnWidthRowHeight_EntirePage_automatic_setting_4()
  With ActiveCell
    .Columns.AutoFit
    .Rows.AutoFit
  End With
End Sub

Sub ColumnWidthRowHeight_EntirePage_automatic_setting_5()
  With ActiveSheet.Range("D6:F115")
    .Columns.EntireColumn.AutoFit
    .Rows.EntireRow.AutoFit
  End With
```

1.11 DEFINING A PRINT AREA, HEADER, AND FOOTER

Our result should not only be visible on the screen, but should also be able to be printed. All the print settings are defined in VBA code using the "**.PageSetup.PrintArea**" function.

We will assume that our table is ready. In VBA code, we first define our print area with the command "**Selection.CurrentRegion.Address**" and assign the variable strArea. Then we wrap them all in the "**.PageSetup**" object with the instruction "**With - End With**".

The special feature here is that the font formatting in the upper or in the lower area are combined with specific texts and entered within quotes with ampersand "&". For example:

.CenterHeader = "&""Arial,Bold""&12O Our Sheet" & Chr(10) & strText & " " & Date

And their alignment for headers and footers is specified as a format code within quotes.

.LeftFooter = **"&8&Z&F"**

At the end, the whole page is displayed in side view with the method **.PrintPreview**.

```
Sub Define_PrintingArea_1()
Dim strPrintingArea As String
Dim strText As String
    strPrintingArea = Selection.CurrentRegion.Address
    strText = "Define Printing Area"
    With ActiveSheet.PageSetup
        .PrintArea = strPrintArea
        .LeftHeader = "Header Align Left"
        .CenterHeader = "&""Arial,Bold""&12Our Spreadsheet" & Chr(10) & strText & " " & Date
        .RightHeader = "Header Align Right"
        .LeftFooter = "&8&Z&F"
        .CenterFooter = "Printed on &D"
        .RightFooter = "&8Footer right &D" & Chr(10) & "&N von &P"
        .CenterHorizontally = True
        .CenterVertically = True
        .Orientation = xlLandscape
        .PaperSize = xlPaperA4
        .FirstPageNumber = xlAutomatic
        .Order = xlDownThenOver
        .FitToPagesWide = 1
        .FitToPagesTall = 1
    End With
    ActiveWindow.SelectedSheets.PrintPreview
End Sub
```

This is how our page will appear:

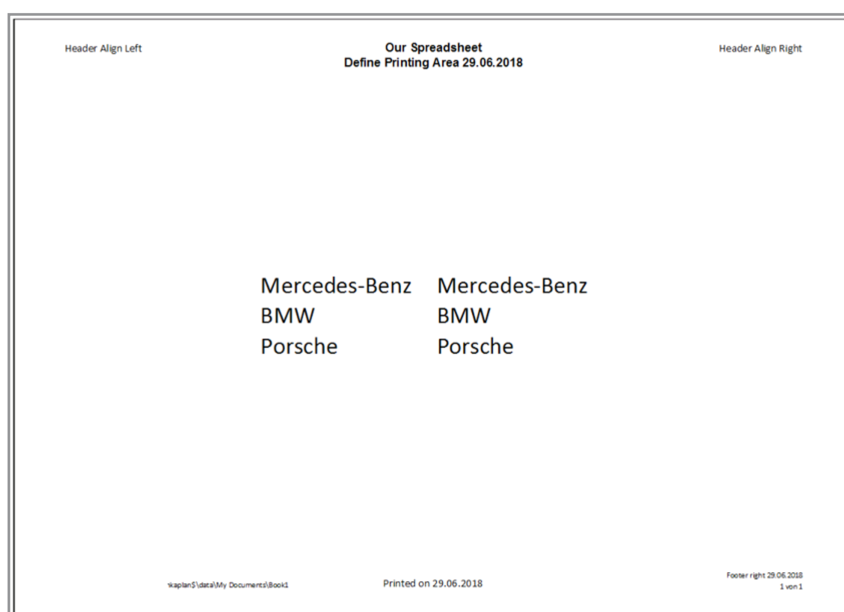


Figure 38: Page view Print Area

All of the important format codes are listed below. Help texts can be found in the "Microsoft VisualBasic Help":

Formatcode	Description
&L	Left-justified „Left"
&C	Centered „Center"
&R	Right-justified „Right"
&D	Current Date „Date"
&T	Current Time „Time"
&F	File Name „File"
&P	Number of Pages
&N	Total number of Pages
&*"Font Type"*	Text inside apostrophes

Table 10: Format code to Print

Although we only used part of the properties of PageSetup in our example, we can see in the remaining properties with descriptions in the table below.

VBA syntax looks like this:
ActiveSheet.PageSetup.*Property*

Setting	Description
Header	
.LeftHeader	Left-justified in header section
.CenterHeader	Middle section in header .
.RightHeader	Right-justified in header section
.LeftHeader	Left-justified in header section
.CenterHeader	Middle section in header
.RightHeader	Right-justified in header section

Download free eBooks at bookboon.com

Setting	Description
Footer	
.LeftFooter	Left-justified in footer section
.CenterFooter	Middle section in footer
.RightFooter	Right-justified in footer section
.InchesToPoints(x.x)	
.LeftMargin	Position from left side
.RightMargin	Position from right side
.TopMargin	Position from top side
.BottumMargin	Position from bottom side
.HeaderMargin	Header from top side
.FooterMargin	Footer from bottom side
.CenterHorizontally	By True horizontal alignment on the print side
.CenterVertically	By True vertical alignment on the print side
.Orientation	„xlLandscape" or „xlPortrait"
Another	
.PrintHeadlings	Printed on True row and column headers
.PrintGridlines	If True, gridlines are printed as well
.PrintComments	Print comments with „xlPrintSheetEnd" / do not print with „xlPrintNoComment"
.PrintQuality	Printquality
.Draft	Print without Chart by True
.Papersize	DINA4 „xlPaperA4"

Table 11: Leading .PageSetup-Property

Download free eBooks at bookboon.com

1.12 AUTOMATIC FILTERING

The automatic filtering "**.AutoFilter**" gives us the possibility to evaluate the data more easily. The AutoFilter character, down arrow, is placed to the right of the cell label. For selected filters, the AutoFilter character is displayed in blue, otherwise in black.

If our table is located in the "A5: C5" area, with the lower VBA code, AutoFilter characters are placed in the fifth line mark in the all filled columns

```
Sub Autofilter_setting() ' Range
 Range("A5:C5").AutoFilter
End Sub

Sub Autofilter_setting() ' Range und Cells
  Range(Cells(5, 1), Cells(5, 3)).AutoFilter
End Sub
```

If we run for the second time, the car filter will be canceled. Now we add the control if filtering exists:

```
Sub Autofilter_control_and_setting()
 If Worksheets("Sheet1").FilterMode = False Then
   Range("A5:C5").AutoFilter
 End If
End Sub
```

1.12.1 QUERIED A AUTOMATIC FILTERING

We learned how to set AutoFilter, queried. Now we reset the result with the method "**.ShowAllData**".

We first ask if any autofilter was used. In this case, all AutoFilters are reset, so make all records visible with "**.ShowAllDate**".

```
Sub ShowAllDate()
  If ActiveSheet.FilterMode Then
    ActiveSheet.ShowAllData
   Else
    MsgBox "Now, all data is visible."
  End If
End Sub
```

Download free eBooks at bookboon.com

1.12.2 SORT WITH AUTOMATIC FILTERING

We can make our filtering dependent on one or more sorting criteria. This sorting is then created with the key **Criterialxy** of the "**.AutoFilter**". Now we see some different examples with lower table.

	A	B	C
1	Obst/Gemüse	Herkunft	Geschmakt
2	Apfel	Deutschland	Sauer
3	Ananas	Ecuador	Süss
4	Birne	Italien	Süss
5	Kiwi	Italien	Sauer
6	Melone	Türkei	Süss
7	Orange	Spanien	Süss
8	Trauben	Türkei	Süss
9	Paprika	Griechenland	Scharf
10	Paprika	Italien	Süss
11			

Figure 39: Example for AutoFilter

Our first example sorts in field 3, so the column "C" after "Sauer":

```
Sub Sortieren_1()
    Selection.AutoFilter Field:=3, Criteria1:="Sauer"
End Sub
```

Our second example sorts in field 1, so the column "A" after the entries starting with "A":

```
Sub Sortieren_2()
    Selection.AutoFilter Field:=1, Criteria1:="=A*", Operator:=xlAnd
End Sub
```

Our third example sorts in field 1 for the entries beginning with "A", in addition to the content "melon:

```
Sub Sortieren_3()
    Selection.AutoFilter Field:=1, Criteria1:="=A*", Operator:=xlOr, Criteria2:="=Melone"
End Sub
```

Our fourth example sorts in field 1 for the entries beginning with "A" and in field 3 for the content "sweet":

```
Sub Sortieren_4()
    Selection.AutoFilter Field:=1, Criteria1:="=A*", Operator:=xlAnd
    Selection.AutoFilter Field:=3, Criteria1:="Süss"
End Sub
```

The sorting can also be dependent on a specific entry:

```
Sub Sortieren_5()
Dim strKriter As String
  strKriter = Range("A1").Value
  If strKriter > 1 Then
      Selection.AutoFilter Field:=3, Criteria1:="Sauer"
  Else
      Selection.AutoFilter Field:=2, Criteria1:="Sauer"
  End If
End Sub
```

We can put together the Criteria1 property. This example sorts in field 1 for the entries that contain "on":

```
Sub Sortieren_4()
Dim strText As String
strText = "an"
   Selection.AutoFilter Field:=1, Criteria1:="=*" & strText & "*", Operator:=xlAnd
End Sub
```

Download free eBooks at bookboon.com

BIBLIOGRAPHY

Online Microsoft Developer Network:

https://msdn.microsoft.com/de-de/vba/office-vba-reference

Download free eBooks at bookboon.com