Excel VBA: Working with Excel Tool Bar Controls

Excel VBA programming - Part 4

Harun Kaplan

bookboon
The eBook company

HARUN KAPLAN

EXCEL VBA: WORKING WITH EXCEL TOOL BAR CONTROLS

EXCEL VBA PROGRAMMING - PART 4

Excel VBA: Working with Excel Tool Bar Controls: Excel VBA programming - Part 4
1st edition

ISBN 978-87-403-2414-3

CONTENTS

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”: Working with Workbooks; Worksheets; Range / Cells; functions; Comments, etc.

In the third book “Excel-VBA Working with Excel Functions & Data”: Handling VBA, own functions and data, graphics, menu creation and Windows Explorer functions with VBA.

In the fourth book “Excel VBA-Working with ToolBar Controls” is mainly about

- Object-oriented programming using UserForm,
- Handling controls of a UserForm.

The target group for the book includes beginners up to the advanced users.

Harun Kaplan

For my Family:
Tülay
Yasin, Sueda, Melik

1 DIALOG SURFACE

Dialog surfaces are forms for the exchange of data between the user, i.e. user, and the computer. We have seen these surfaces previously in the topics MsgBox or InputBox.
For example:

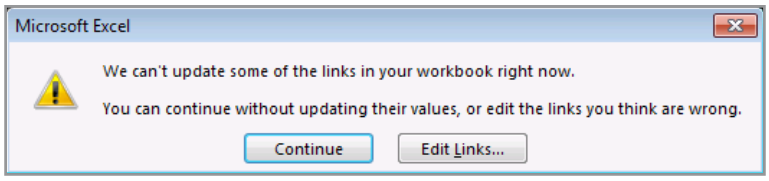


Figure 1: Example an Error message

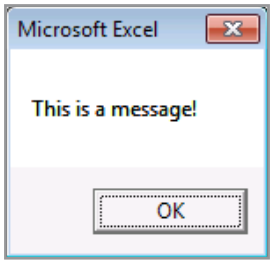


Figure 2: Example a message

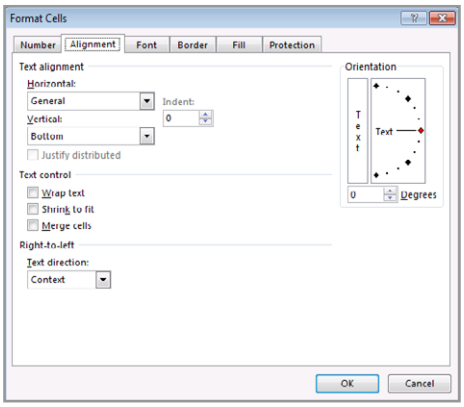


Figure 3: Setting a cell

They contain at least one or more controls. For example, buttons like “OK”; “Abort, stop”; Radio buttons; Dropdown fields, etc.

We can either insert these controls directly into the table or into the UserForm interface. To insert a control into the table, we open the menu “Developer Tools / Insert”, then we select an element.

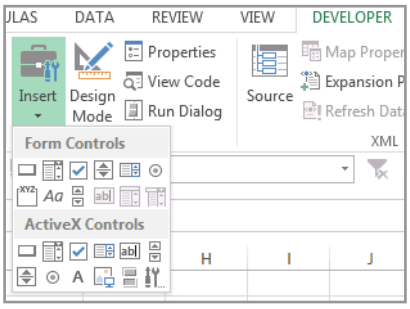


Figure 4: Insert a control in Excel sheet

The form controls are for tables and ActiveX controls are for both the tables and UserForm surfaces. The only difference is that the terms of the form controls are in German, e.g. "Button1" and ActiveX elements in English, e.g. "CommandButton1" are.

First we will put our entries directly into the table.

1.1 CONTROLS IN EXCEL SHEET

Form controls:

- We can embed it directly in the respective table and with German naming,
- Recorded / written procedure is stored in the table module.

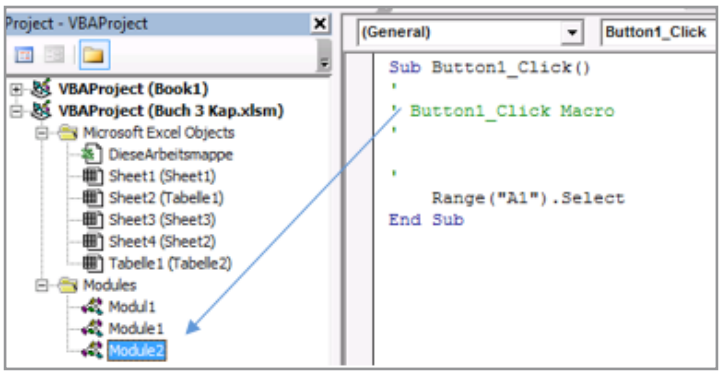


Figure 5 : Inserted Button with his procedure

ActiveX controls:

- We can embed it directly in the respective table and with English name
- Recorded / written procedure is stored in the respective table.

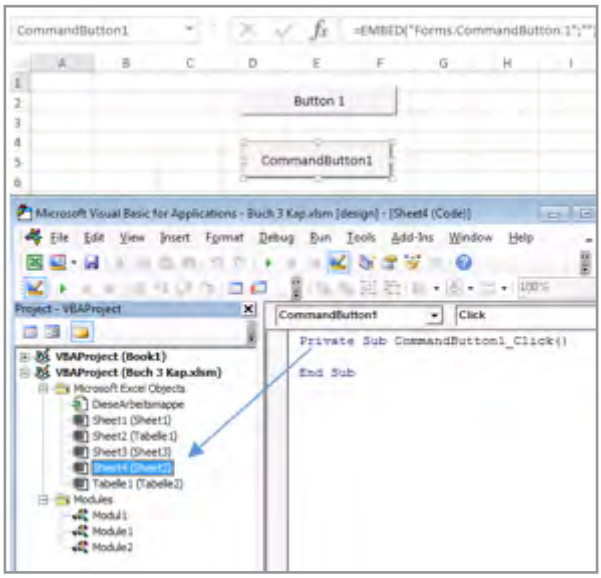


Figure 6: ActiveX-Control in Sheet1(Sheet 1)

In the toolbar "Controls-Toolbox" we see more icons.

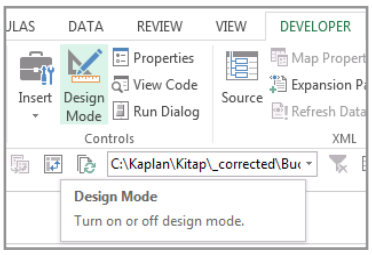


Figure 7: Icon's Controls-Toolbox

We'll talk to you briefly:

-  Design Mode

After creating a control, draft mode is automatically activated. A double-click on the control opens the corresponding module of the worksheet. To use the control in the worksheet you must exit Design mode by clicking the Design Mode icon.

-  Properties

Each control has different properties.
The assignment or specification of the properties is carried out:

- using the VBA code,
- via the properties window.

The properties are only active in the drafting mode.

-  View Code

Instead of double-clicking on the control menu, we can also access the editor via “Display Code”.

-  Run Dialog

UserForm can be executed with the “Execute dialog box”. Later we see this point in detail.

1.1.1 INSERT CONTROL ELEMENT IN SHEET

In our example, we have selected a command button. If the Properties window is not visible, right-click on the element and select Properties

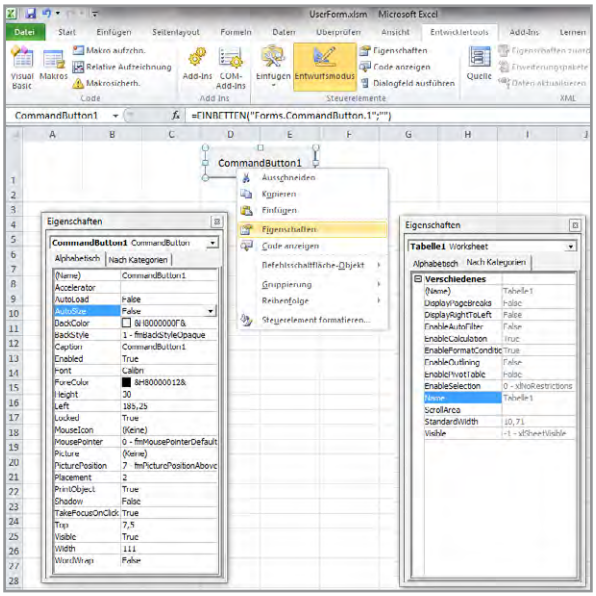


Figure 8: Properties alphabetically / category

The controls in the Properties window can be sorted either alphabetically or by category. The categories can be shown or hidden with the minus or plus sign. Depending on the type of control, it can either be edited or selected from the offered key.

There are eight anchor points visible on our inserted control. They serve to change the size of the element.
A control element can be more precisely embedded by holding the Alt-key over the cell area.

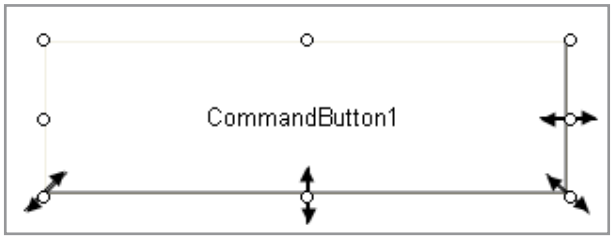


Figure 9: Change size und position of the elements

- With this arrow ↔ the selected element can be changed in the direction of the arrow.
- By holding the left mouse button the element can be positioned as free-floating

The numbers of the elements are incremented the next time a similar element is inserted. For example, CommandButton1, CommandButton2, CommandButton3, and so on.

The extensive procedures are very easy to spin. That's why we should work with a so-called prefix and a meaningful name. A prefix is an abbreviation of the English term for a control. Our prefix in the above example is cmd. This is an abbreviation of CommandButton.
We covered prefixes in detail in the first book.

CommandButton is a frequently used control. We insert a button in the table and then make some changes in the properties window. By double-clicking the button, we open the VBA and store the procedure call. The "Click" event is already displayed with the procedure name.

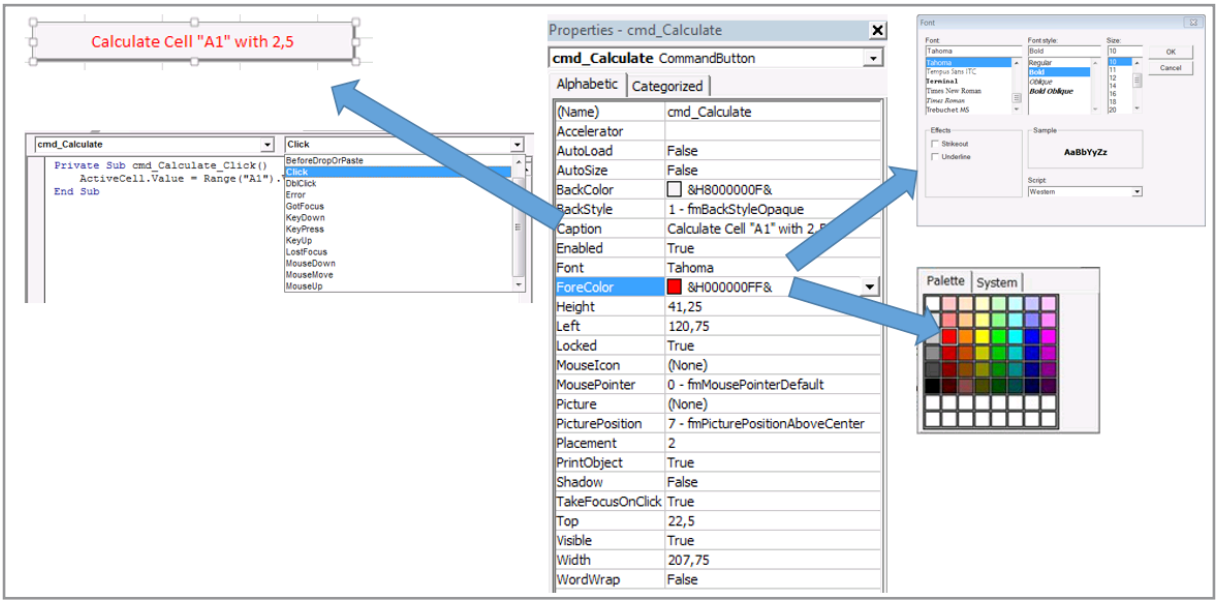


Figure 10: Changes / Element Denotations

In this example we see:

- the composition of the procedure naming (consists of the element name with event procedure),
- the setting options in the properties window.

The short example should multiply the value 2.5 by the content of the cell "A1". The result should be entered in the active cell.

Variante 1:

Without using a subroutine:

```
Private Sub cmd_Calculate_Click()
        ActiveCell.Value = Range("A1").Value * 2.5
End Sub
```

Variante 2:

Calling a subprogram with the "Call" command:

```
Private Sub cmd_Calculate_Click()
        Call Calculate
End Sub
```

Here is the sub-procedure:

```
Sub Calculate()
        ActiveCell.Value = Range("A1").Value * 2.5
End Sub
```

1.2 DIALOG SURFACE „USERFORM"

When we work with the Office package, we are often unknowingly using UserForms. For example, the "Options" window from the "File" menu. They are simple selections; Radio buttons; Selected objects; Check box; Identification fields; Combo box; Register or multi-pages, etc. to recognize.

The design of a user form is nothing more than a simplification and condition of a procedure. With these UserForms we "speak" with the Excel and tell it what and how something should be done for us.

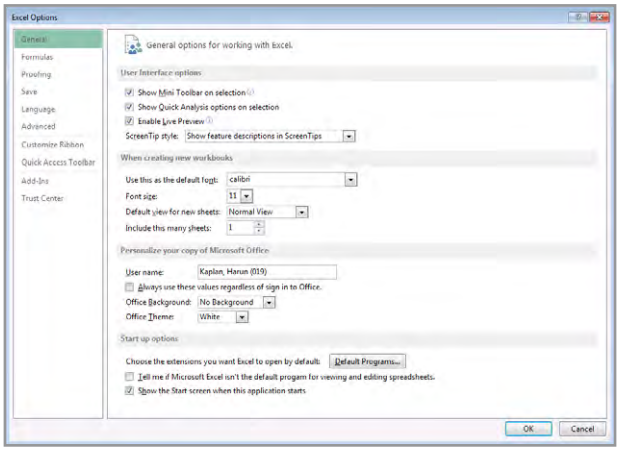


Figure 11: An Example UserForm "Excel-Options"

- It is very useful for forms to enter the data of a table or a database in a simpler and clearer way. Well, for our controls we need a UserForm interface. We can open the services with
- the VBA menu "**Insert | UserForm**"

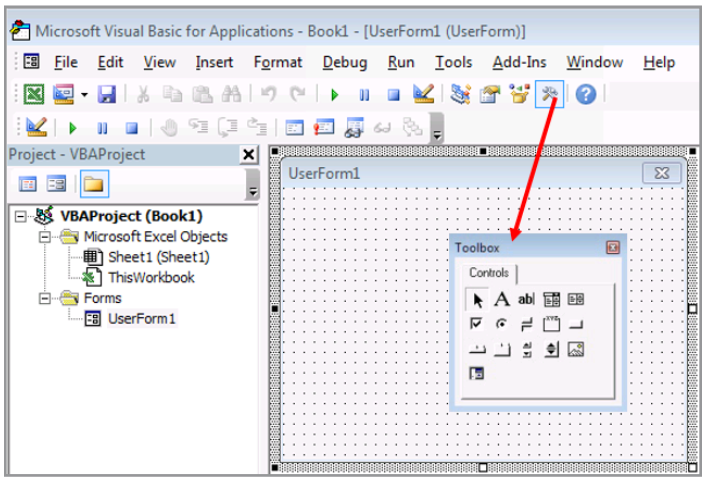


Figure 12: Adding UserForm about VBA-Menü

- directly above the "Insert UserForm" icon in the "Preset" toolbar.

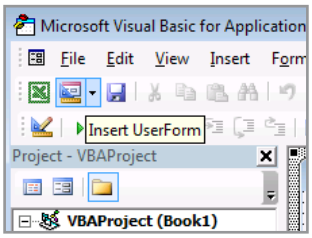


Figure 13: Adding "Insert UserForm" per Icon

The toolbox contains all the elements of the Controls Toolbox plus the Frame, Register, Multisite, and RefEdit elements.

After the request, VBA will provide us with an empty UserForm interface. This can be reduced or enlarged as required. We can create our form with the controls from the toolkit. The handling of these elements is no different than those in the Controls Toolbox. In other words, we can use the mouse to adjust our settings, such as font type, size and color, mouse positions and sizes, or enter them directly in the Properties window.

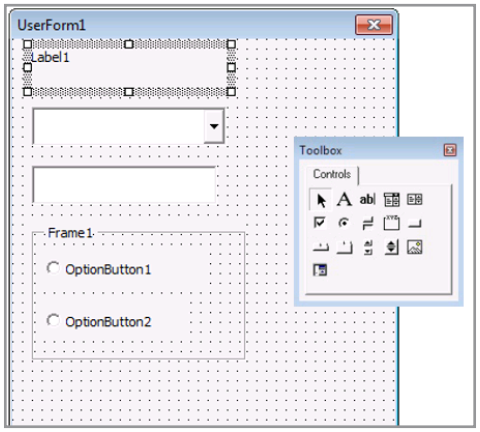


Figure 14: UserForm Elements

Before we get to know the individual elements of the toolbar of the UserForm, let's take a look at the "Standard" toolbar.

Figure 15: Toolbar "Standard"

With the first Excel symbol we change the level Excel | VBA alternately. The disk icon saves our VBA. The middle symbol is the insertion of "UserForm" or "Module" or "Class Module".

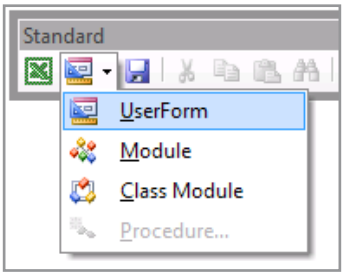


Figure 16: Adding UserForm | Modul | Class Module

With the next three symbols we can start / stop or stop a VBA procedure.

Figure 17: Run Sub / UserForm | Break | Reset

On the right side there are another five symbols.

1. The design mode is used to select controls

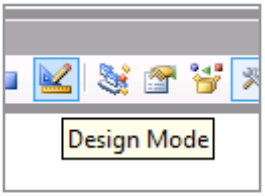


Figure 18: Call Design mode

2. The Project Explorer is used to invoke the "Project VBAProject" window. Thus, we have the opportunity to see, among other things, where I am.

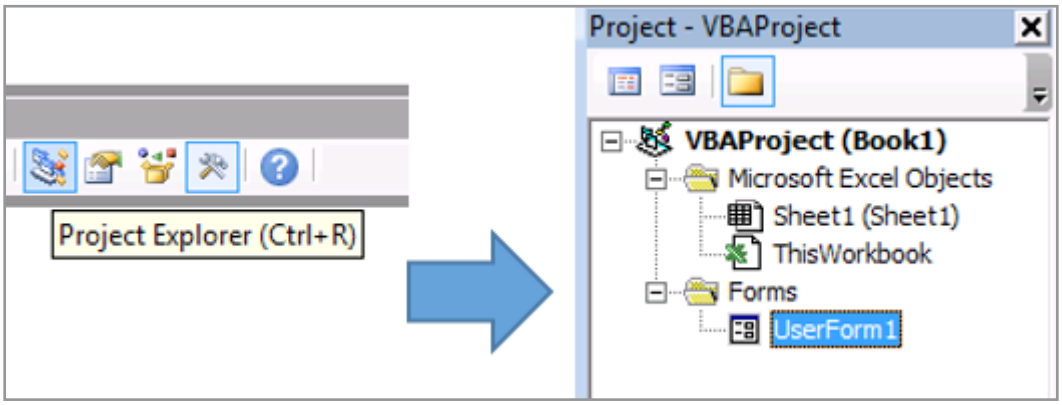


Figure 19: Call Project Explorer

3. The Properties window can be accessed either via this icon or "F4" key. This window is used very often because it allows us to individually adapt or format our elements. We will see that later in greater detail.

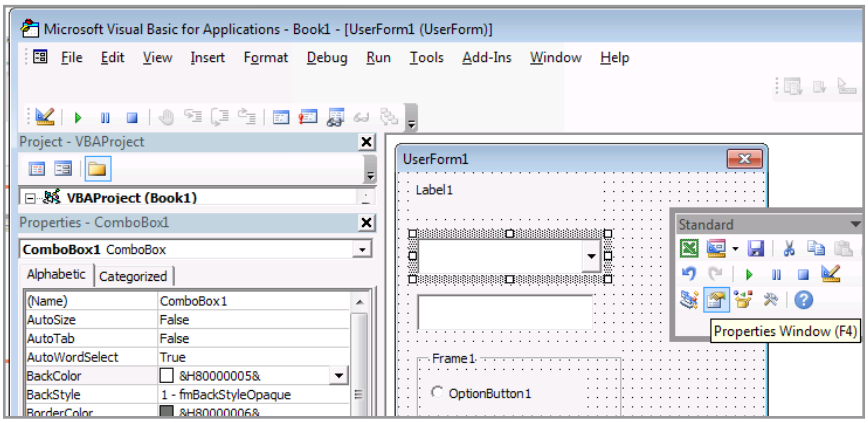


Figure 20: Call Properties Window

4. The object catalog contains all VBA codes. Even those which have no help texts. We can search for the terms here, see contexts, etc.

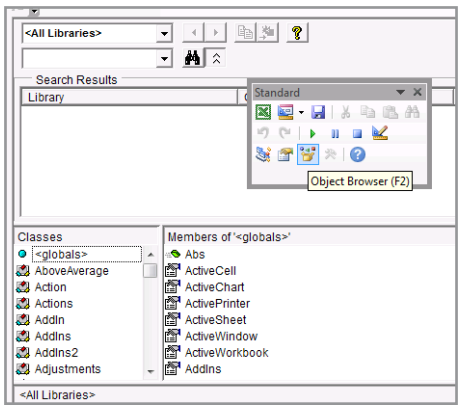


Figure 21: Call Object Browser

5. The toolkit is the most important icon. There we can find all UserForm elements.

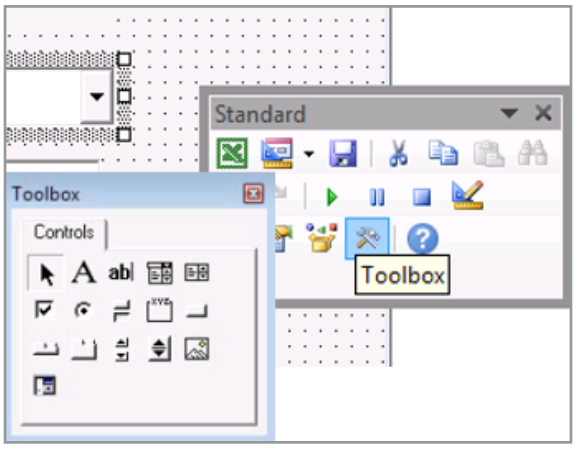


Figure 22: Call Toolbox

Before we tackle the detailed information of the elements of the toolkit individually, we will first look at the following points:

- Collection tool bar,
- Properties,
- Aligning an element,
- User form bar,
- Activation order,
- Starting, closing, _Initialize, _Activate or Change size and position of an UserForm,
- Control activity or visibility change,
- Prevent "Ending over red x" (at the left upper edge).

1.2.1 PROPERTIES WINDOW

The properties window  is visible in the icon. As mentioned earlier, each control object can be accessed through the Properties window. The list of contents can be sorted alphabetically or grouped by category. Status, size, position, formatting, etc. can be entered directly here.

There are two ways to access the Properties window. We select a control and

- click on the symbol Properties window
- use the F4 key.

In the left column are the properties whose current values can be seen in the right column. Predefined data in the right-hand column can be changed either with direct input or with the desired value or option.

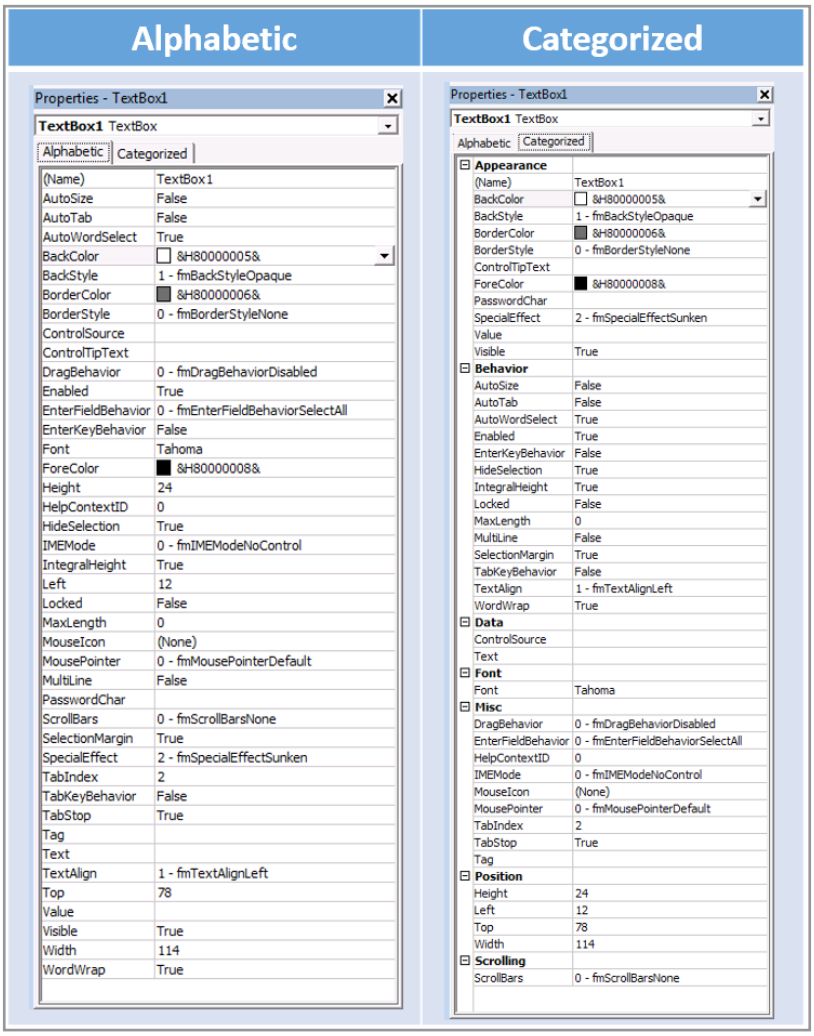


Figure 23: Properties Window sorted

If we want to customize the control of many elements on UserForm, we can open our pull-down list box and select the desired element. Then a change can be made.

As an example we will select a button here, then ....

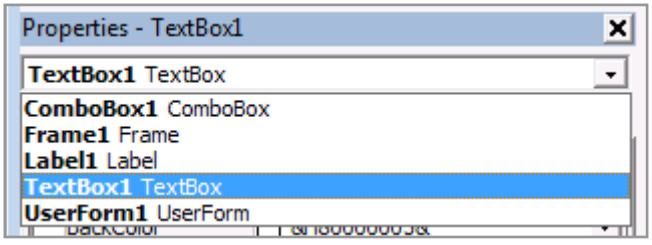


Figure 24: Select a control from more

... we adapt the following properties to the CommondButton element in VBA as follows:

The appearance of a control:
BackColor *vbYellow*
Caption name of controls *Search*
ForeColor Font color *vbBlue*

The Position of a control:
Height on *40*
Left distance to left side of UserFrom *156*
Top distance to top side of UserFrom *30*
Width on *60*

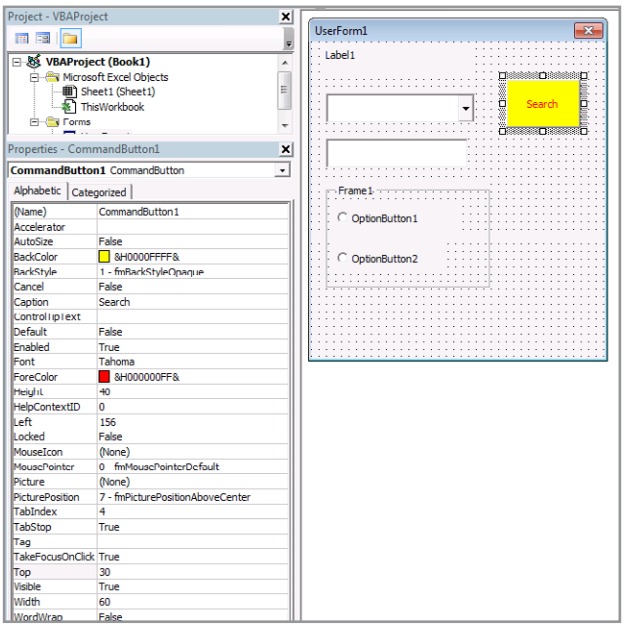


Figure 25: Setting in properties window

With this listing, all settings are displayed immediately after calling the UserForm interface:

```
Private Sub UserForm_Initialize()
  With cmdSuchen
        .Caption="Search"
        .BackColor = vbYellow
        .ForeColor = vbRed
        .Height = 40
        .Left = 156
        .Top = 30
        .Width = 60
  End With
End Sub
```

1.2.2 ADJUSTING USERFORM CONTROLS

The raster-units in the width and height of a UserForm are set by default to six points. This setting can be changed as needed in the VBA Tools / Options menu on the General tab. This options window is also an example of a UserForm with many UserForm elements.

We can activate or deactivate further settings according to our needs.

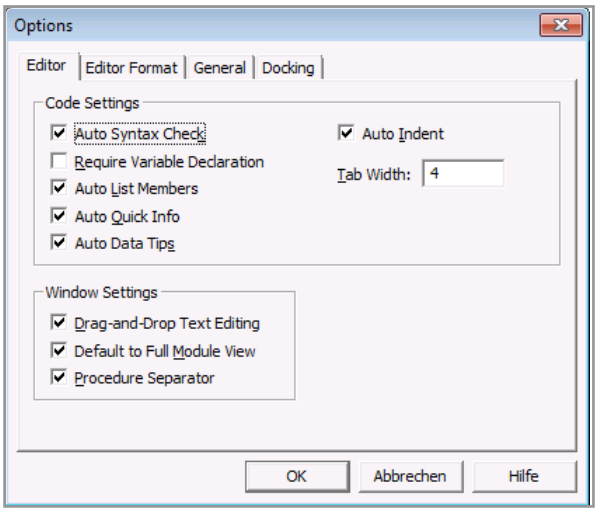


Figure 26: About VBA-Menü "Tools | Options"

If we have created one or more elements in our UserForm, we can align them in the Format menu depending on the selection. To mark an element, we simply click on the element. The marked element is framed with eight anchor points.

The selection of the several elements can be carried out by holding down the Ctrl key and using the left mouse button.

Or by dragging the elements while the left mouse button is pressed. All the elements touched by the mouse cursor are selected.

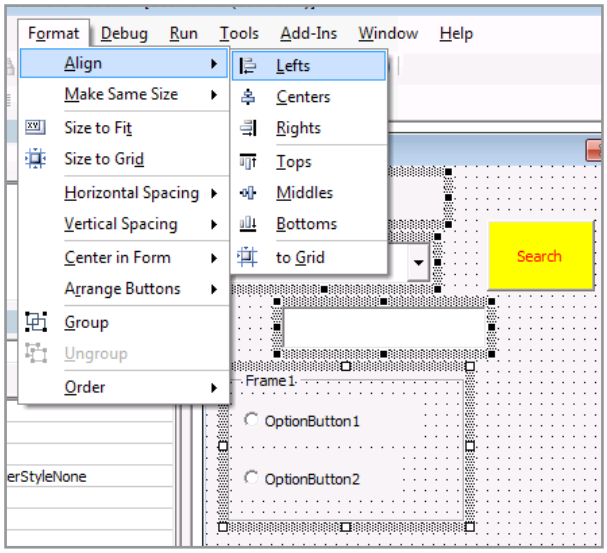


Figure 27: Adjusting UserForm controls

We will not cover all formatting options here individually. Most symbols are self-explanatory. Learning by doing is very helpful here.

But we should briefly look at the three most important ones:

Alignment to the left: Alignment of the left-most element of selected elements. In our example above to CheckBox1.

Match Size Width: Match by smallest item of selected items. In our example above it is shown as Label1.
Group: selected elements are used to form a group.

1.2.3 TOOLBAR USERFORM

For quick alignment or positioning of controls, we can also get the toolbar of the UserForm for help.

Figure 28: Toolbar UserForm

If it is not visible, it can easily be selected by right-clicking on the toolbar and UserForm.

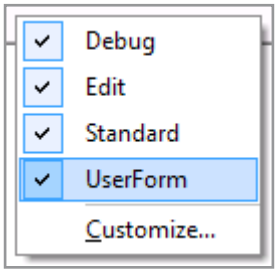


Figure 29: Select UserForm-Toolbar

Here we find most of the format functions of a control. As long as no element is selected, our bar remains inactive (Figure 31). Only when at least one element has been selected, recognizable by the framing of the element, is our bar active (Figure 32).

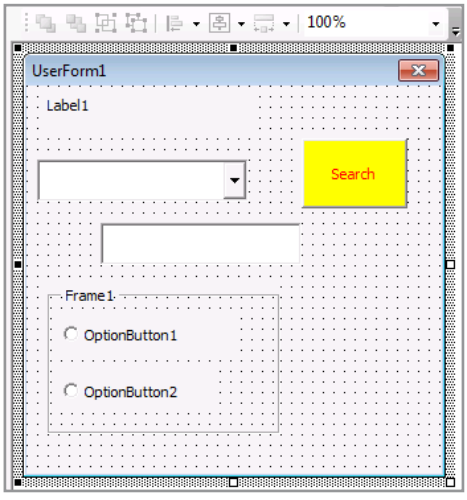


Figure 30: Toolbar Userform inactive!

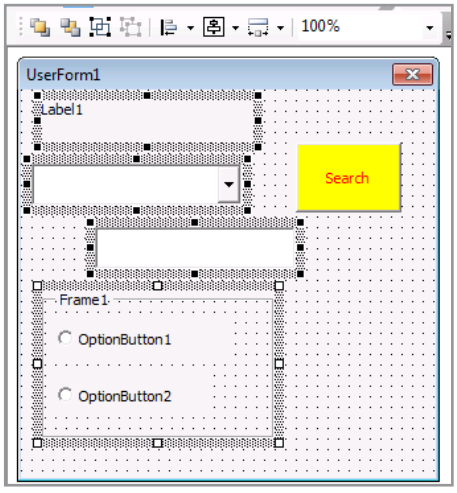


Figure 31: Toolbar Userform active!

1.2.4 SET THE ORDER OF TOP

The order of the controls is automatically set in UserForm after they have been inserted. Figure 33 contains three elements. First a CheckBox is inserted, then an OptionsButton and at the end a TextBox is inserted. This order stays the same unless it is manually changed.

It would be very complicated to use the UserForm if the order of the controls were undefined. Working with a UserForm with extensive controls without ordering would be quite inconvenient.

If we jump from one field to the next with the tab key, the cursor may jump back and forth across the page. That's not ergonomic.

In a UserForm with sequence-defined controls, we can easily jump back and forth with the Tab key to the next item or with keyboard shortcut **Ctrl + Tab keys** in the reverse order.

To change the activation order, we first activate the UserForm in design mode and then select the menu command "**View | Top Order**".

Show all inserted controls as we have inserted them. To change an order of a control, select an element and rearrange it at the desired position using the "**Move Up**" or "**Move Down**" buttons.

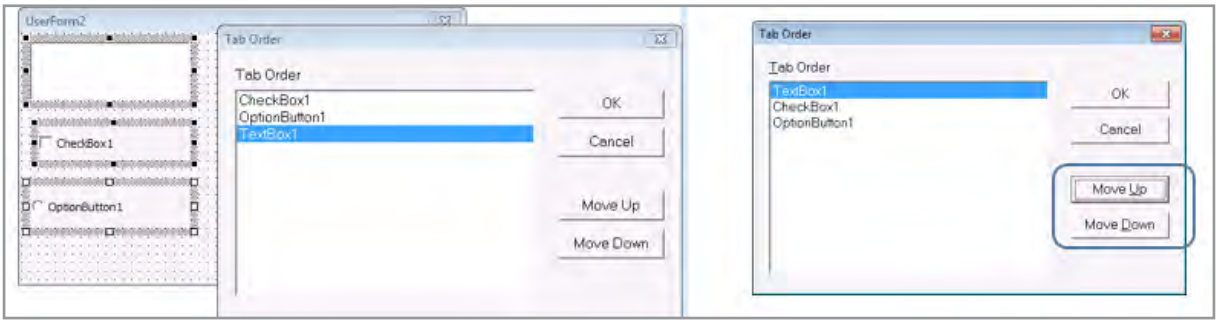


Figure 32: Top order: before / after

1.2.5 DISPLAYING AND CLOSING A USERFORM

Before we start with the individual buttons of the dialog, we will learn how to open and exit the UserForm mask. We open it using the "**.Show**" method and exit it with the "**.Unload**" statement.

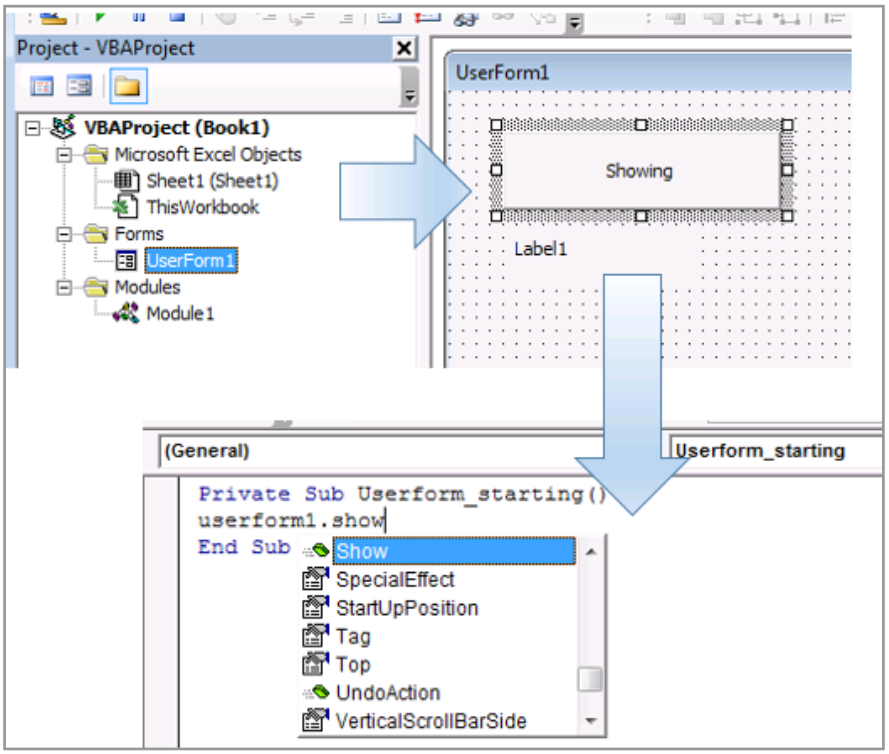


Figure 33: Displaying a UserForm

To open the UserForm automatically after starting the file, the startup can be entered in the object "ThisWorkbook" with Workbook_Open.

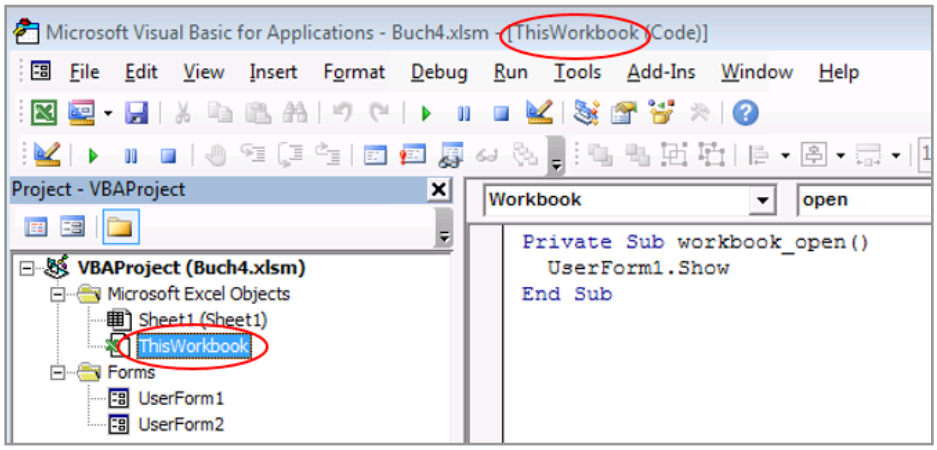


Figure 34: Workbook_Open in ThisWorkbook

An Excel spreadsheet can't be edited while the UserForm is active. It remains locked. There are three ways to edit a table while the UserForm is active:

➢ Unlocking the table: change the setting in the properties window to ShowModal = False.

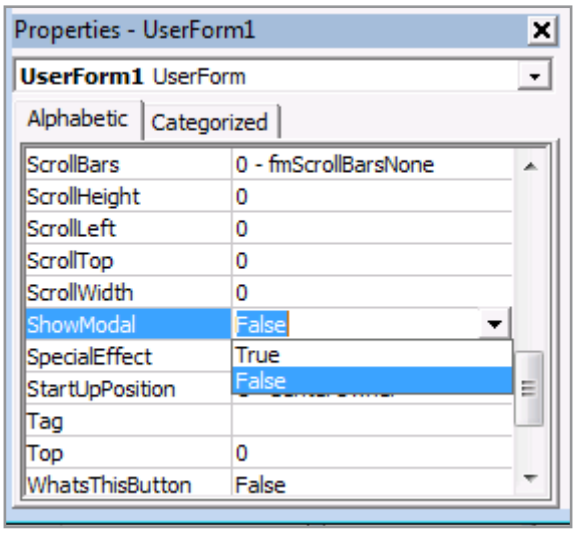


Figure 35: Setting **ShowModal** on **False**

➢ enter the constant vbModeless in VBA code when opening the UserForm

```
Private Sub Workbook_Open()
        UserForm1.Show vbModeless
End Sub
```

➢ open the Show method with the value 0 in VBA code when opening the UserForm

```
Private Sub Workbook_Open()
        UserForm1.Show  0
End Sub
```

Well, everything that was started will also eventually end. ☺. This, too. A UserForm mask can be terminated with the Unload statement.

```
Private Sub cmdEnding_Click()
   Unload UserForm1
End Sub
```

1.2.6 USERFORM_INITIALIZE()

UserForm_Initilaize() is similar to auto_open(). All VBA codes entered here are executed after calling the UserForm. All procedures and UserForm procedures are written to the UserForm level:

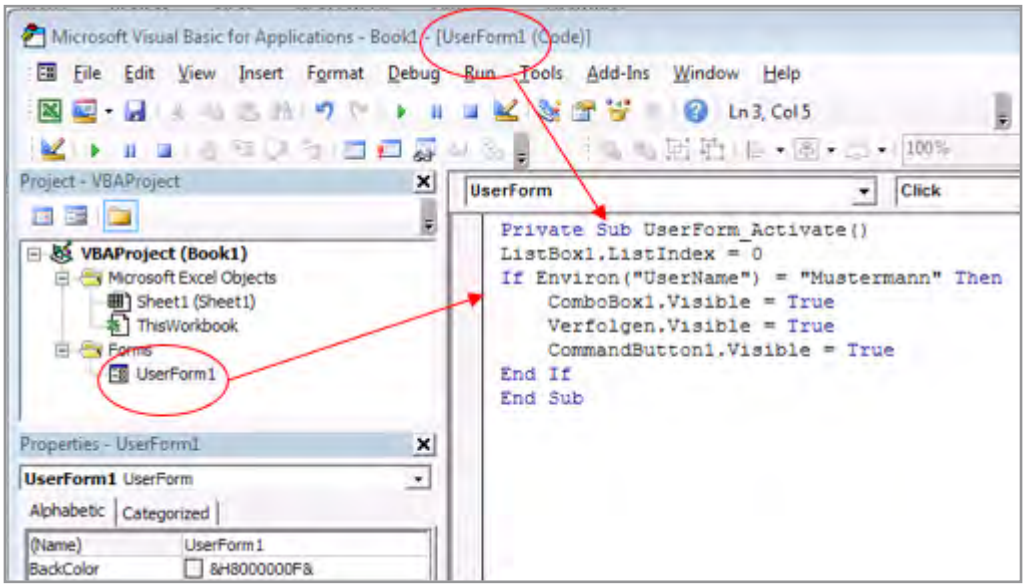


Figure 36: On level UserForm

In UserForm_Initialize () all necessary basic settings can be entered. In the next example, listed items are only visible when this procedure has been started by the "Mustermann" computer.

```
Private Sub UserForm_Initialize()
ListBox1.ListIndex = 0
 If Environ("UserName") = "MuMann" Or Environ("UserName") = "MUSTERMANN" Then
    ComboBox1.Visible = True
    Verfolgen.Visible = True
    CommandButton1.Visible = True
End If
End Sub
```

1.2.7 USERFORM_ACTIVATE()

UserForm_Activate (), on the other hand, runs after each activation of the UserForm.

```
Private Sub UserForm_Activate()
ListBox1.ListIndex = 0
 If Environ("UserName") = "MuMann" Or Environ("UserName") = "MUSTERMANN" Then
    ComboBox1.Visible = True
    Verfolgen.Visible = True
    CommandButton1.Visible = True
End If
End Sub
```

1.2.8 SETTING USERFORM SIZE

By default, when we create a new UserForm window, the width is 330 and the height is 248.25 pixels. This window can be automatically adjusted to the entire size of the Excel window as desired. For our example, we zoom out our Excel window so that is only positioned in the middle of the monitor.

We used a term "**Me**" here. This is the current top-level UserForm window. This window can be abbreviated as "Me" in UserForm VBA.

```
Private Sub UserForm_Activate()
  With Me
    .Height = Application.Height
    .Width = Application.Width
  End With
End Sub
```

Well, we resized our Excel window as described earlier. Then our result looks like this:

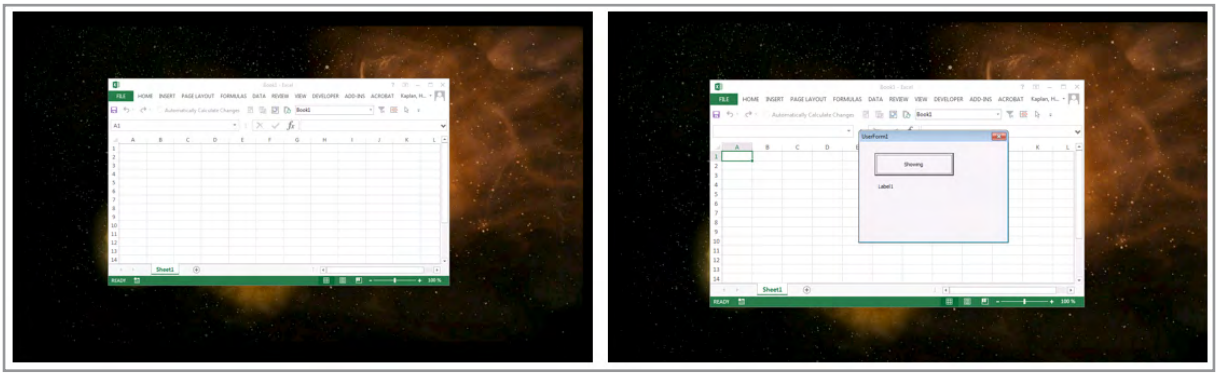


Figure 37: Size of UserForm on Excel window

We can also specify the sizes directly:

```
Private Sub UserForm_Activate()
  With Me
    .Height = 400
    .Width = 500
  End With
End Sub
```

If we want it to be full-sized, we must first bring our Excel window to full size. Here we reach with Application.WindowState = xlMaximized. Our listing now looks like this:

```
Private Sub UserForm_Activate()
Application.WindowState = xlMaximized
With Me
    .Height = Application.Height
    .Width = Application.Width
  End With
End Sub
```

1.2.9 POSITIONING USERFORM

We can also position our UserForm window in the original size on the Excel interface.

With the properties

- **.Top**, the distance from the top edge
- **.Left**, the distance to the left edge

of a UserForm window.

```
Private Sub UserForm_Activate()
Application.WindowState = xlMaximized
With Me
    .Height = 400
    .Width = 500
    .Top = 100
    .Left = 50
  End With
End Sub
```

Or we put the active UserForm window in the middle of the Excel interface. By combining the above- mentioned sizes and positioning properties as a formula:

```
Private Sub UserForm_Activate()
Application.WindowState = xlMaximized
  Me.Left = (Application.Width - Me.Width) / 2
  Me.Top = (Application.Height - Me.Height) / 2
End Sub
```

1.2.10 VISIBILITY OF TOOLBOX CONTROL

Most controls are set by default to the properties "**Visible**" and "**Enabled**".

The property **.Enabled = False** will cause each item to not respond to triggered activity. In other words, it does not work.

The property **.Visabled = False** makes the respective element invisible.

```
Private Sub UserForm_Activate()
  CommandButton1.Visible = False
  TextBox5.Enabled = False
End Sub
```

1.2.11 BREAK UP A USERFORM

A UserForm window can be stopped in two ways:

1. Using a programmed control:

```
Private Sub CommandButton1_Click()
  Unload Me
End Sub
```

2. Via the red "**X-sign**" in the upper right corner.

The red "X-character" can be deactivated and the termination can be done via the programmed control. To disable the "X-character" we will write a class called "QueryClose". The necessary definitions, Cancel and CloseMode, are already entered in parentheses. These are also necessary.

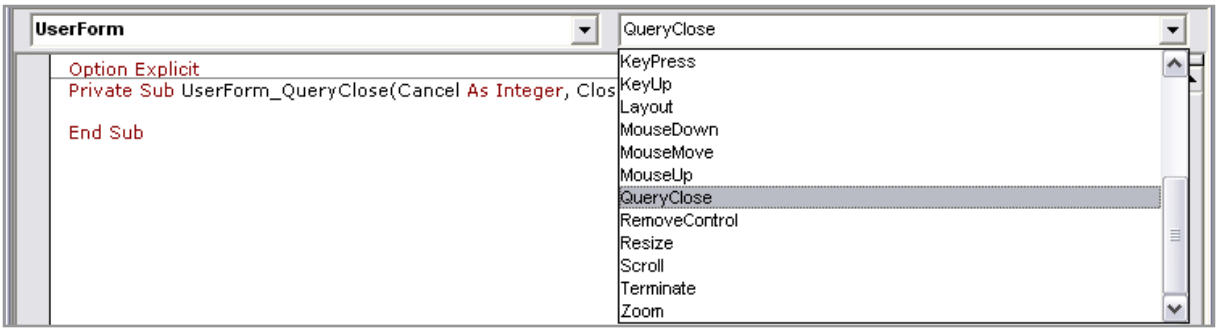


Figure 38: VBA-Code for UserForm_QueryClose

Then we can check the locking mechanism with an "if query" to see if a button has not been clicked. If this is the case a message will be displayed.

```
Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
    If CloseMode <> 1 Then
        Cancel = 1 'or with Cancel = True
    MsgBox ("It only ends with CANCEL button.")
    End If
End Sub
```

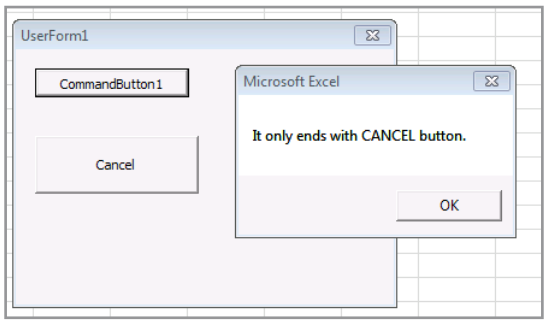


Figure 39: Result UserForm_QueryClose

Now we can finally come to the individual elements.

1.3 TOOLBOX CONTROLS OF USERFORM

As mentioned earlier, the controls are graphic objects. These can be placed on an Excel table or a UserForm. We covered these elements in the previous chapter. They facilitate data entry. Well, let's get to know the individual elements with the possible applications. We will build our examples with previous elements so that we can use them as quickly as possible in practice.

The order of the elements as sorted in the toolbar is already set up.

I think we can start with the first control now:

1.3.1 LABEL

A **Label** can be recognized by the symbol  . These fields can also be used as a visible "value field" or as a "button". This means that a non-editable result or information can be output.

If we have inserted a label in our UserForm, this element automatically carries the name "Name: Label1". Its caption text is "Caption: Label1". As already mentioned, these can be changed either in the properties window or via the VBA code.

Well, let's look at an example:

In the cell "A1" there is a value "12" and in the cell "A2" as well as the value "21". These values should be summed together.

For this we create a UserForm mask. We manually format the "Label1" to "Label3" in the properties window and "Label4" to "Label6" are formatted by VBA code. In our example we see both variants.

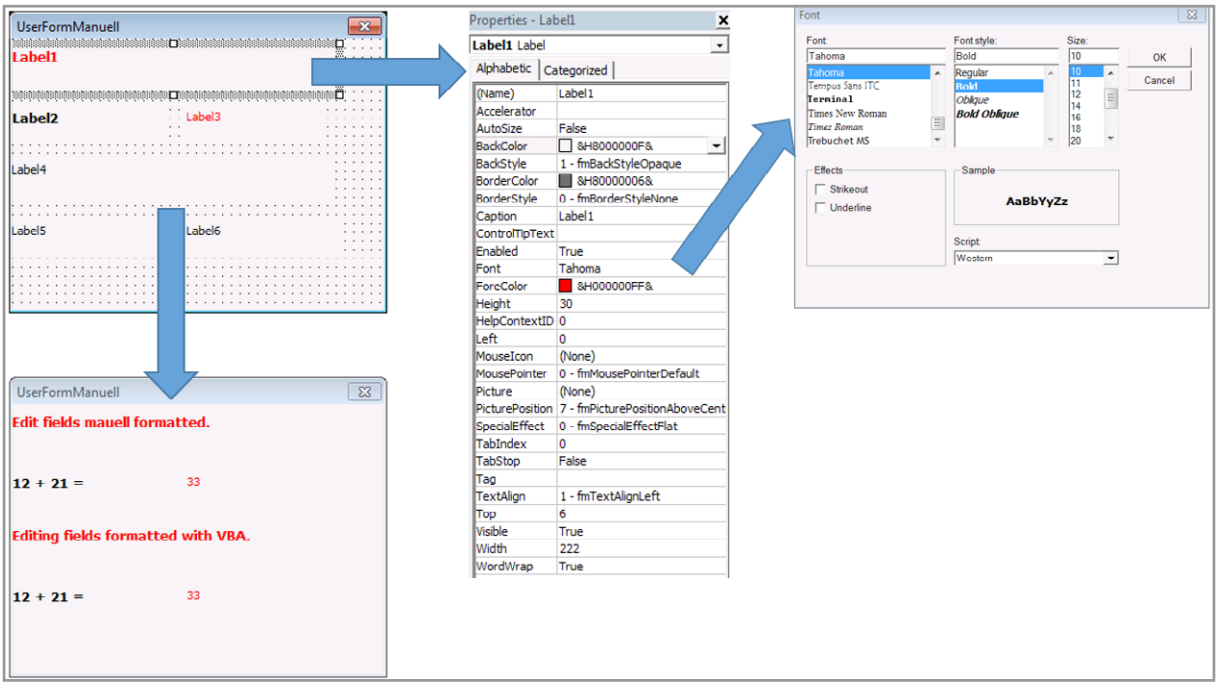


Figure 40: Setting the Input controls

```
Private Sub UserForm_Activate()
  'Manually formatted
  Label1.Caption = "Manually formatting a label"
  Label2.Caption = Range("A1").Value & " + " & Range("A2").Value & " = "
  Label3.Caption = Range("A1").Value + Range("A2").Value

  'Formatting abaut VBA-Code
  With Label4
    .Caption = " VBA formatting a label"
    .ForeColor = RGB(255, 0, 0) 'red colour with RGB or with vbRed
    .Font.Bold = True
    .Font.Size = 10
  End With
    With Label5
    .Caption = Range("A1").Value & " + " & Range("A2").Value & " = "
    .Font.Bold = True
    .Font.Size = 10
    .Width = 100
    .Height = 24
    End With
  With Label6
    .Caption = Range("A1").Value + Range("A2").Value
    .Font = "Tahoma"
    .Width = 220
    .Height = 24
    .ForeColor = vbRed 'red colour with Colour-Index
  End With
End Sub

Private Sub Label7_Click() 'starting a file
   Workbooks.Open ("C:\Kaplan\Autos.xlsx")
End Sub
```

1.3.2 TEXTBOX

A **TextBox** can be recognized by the symbol  . This is like a label, but with an input option. This means that this field can be edited manually, similar to an InputBox.

After pasting into UserForm and double-clicking a text box, we enter the VBA code entry window with a "_Change" event.

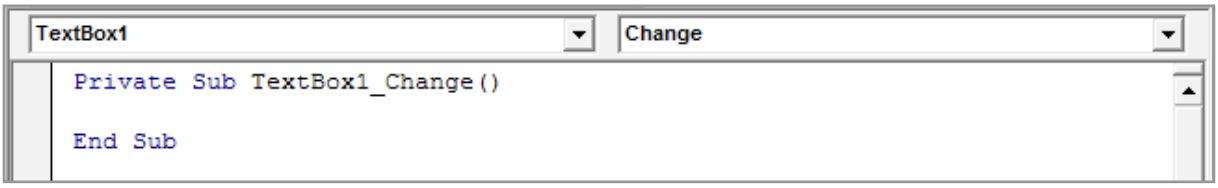


Figure 41: TextBox1_Change() in VBA-Editor

In a text field, visible or invisible characters can be entered. For example, when entering passwords invisible characters are displayed as an asterisk, or another character is displayed to represent the invisible character entered. For additional settings we either use the properties window or we enter directly into procedure.

The text box is usually single-line. If we want to enter multiple lines, we need the properties "**MultiLine**" and "**WordWrap**". These properties are used together.

- **.Multiline = True**
- **.WordWrap = False**

The length of an input in the text field can be restricted with the property "**MaxLength**". .MaxLength = 5

Now for our example:

The content of "TextBox1" should be entered in cell "A2" and the content of "TextBox2" should be entered in cell "B2". The special feature of "TextBox2" is that there is no text here, so only numbers can be entered.

This query is done with VBA code. The "TextBox3" has a fixed content and should be entered in cell "C2". To do this we set the key in the properties window of "Enabled = False".

If we want to make a field permanently non-editable, then a label is useful.

Lastly, "TextBox4" content should fill the cell "D2" and this should be multi-line. When editing, we can use the key combination "Shift + Enter" to insert a line break.

"TextBox5" is defined as a Password field using the **.PasswordChar** key. Each character should show a star. In the cell "E2" we can still see the entered value.

Well, that's our listing:

```
Private Sub Userform_Activate()
  With TextBox3
    .Value = Range("C2").Value
  End With
  '
  With TextBox4
    .MultiLine = True
    .Value = Range("D2").Value
  End With
'
  With TextBox5
    .PasswordChar = "*"
  End With
End Sub

Private Sub TextBox1_Change()
Range("A2").Value = TextBox1.Value
End Sub

Private Sub TextBox2_Change()
  If IsNumeric(TextBox2.Value) Then
    Range("B2").Value = CDbl(TextBox2.Value)
  Else 'In case of incorrect entry
    MsgBox "How old are you?  Pleas, enter as value! "
  End If
End Sub

Private Sub TextBox3_Change()
  Worksheets("Sheet1").Range("C2").Value = TextBox3.Value
End Sub

Private Sub TextBox4_Change()
  Worksheets("Sheet1").Range("D2").Value = TextBox4.Value
End Sub

Private Sub TextBox5_Change()
  Worksheets("Sheet1").Range("E2").Value = TextBox5.Value
End Sub
```

And here is the result:

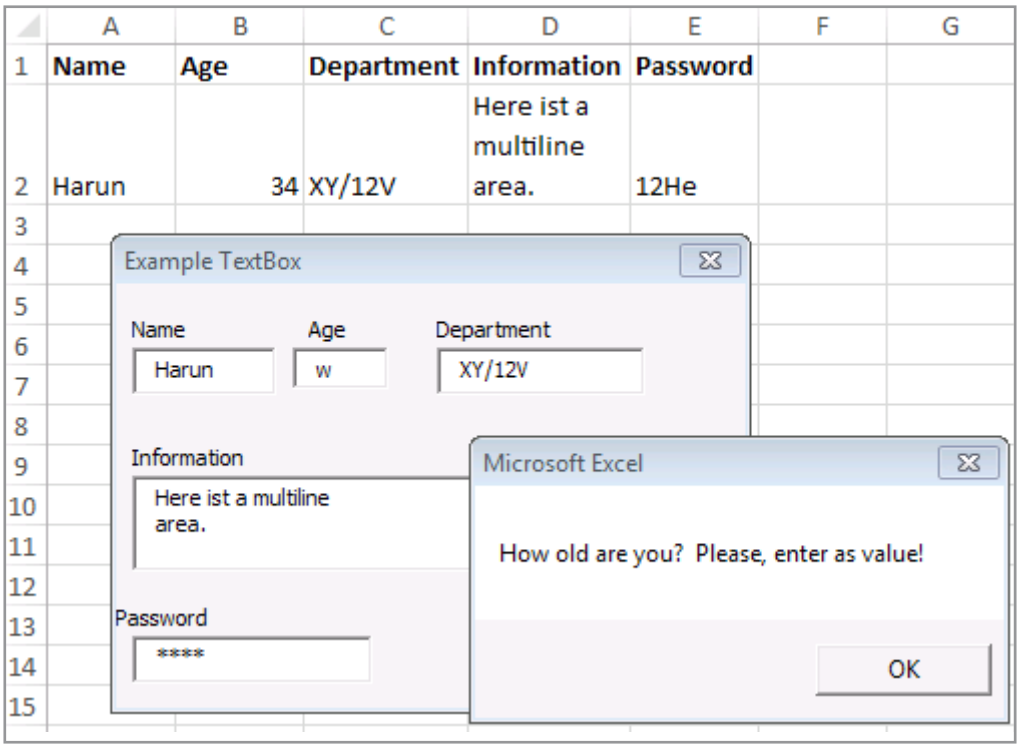


Figure 42: Differently formatted text fields

1.3.3 COMMANDBUTTON

A **CommandButton** can be recognized by the symbol  . A UserForm has at least one command button. Be it to exit, to save the entered data or information or to call another UserForm.

We can take the previous example and add two more buttons. One of the buttons enters the entered data in the table as the last entry and the second button is used to empty the input fields, i.e. the text boxes.

```
Private Sub CommandButton1_Click()
Cells(Rows.Count, 1).End(xlUp).Offset(1, 0).Select
  If IsNumeric(TextBox2.Value) Then
    ActiveCell.Offset(0, 0).Value = TextBox1.Value
    ActiveCell.Offset(0, 1).Value = CDbl(TextBox2.Value)
    ActiveCell.Offset(0, 2).Value = TextBox3.Value
    ActiveCell.Offset(0, 3).Value = TextBox4.Value
  Else
    MsgBox "How old are you?  Please, enter a value! "
    Me.Show
  End If
End Sub

Private Sub CommandButton2_Click()
'Empty TextBoxes if you make a mistake or enter something else
  TextBox1.Value = ""
  TextBox2.Value = ""
  TextBox3.Value = ""
  TextBox4.Value = ""
  TextBox5.Value = ""
End Sub

Private Sub Userform_Activate()
Worksheets("Sheet1").Select
  With TextBox4
    .MultiLine = True
  End With
End Sub
```

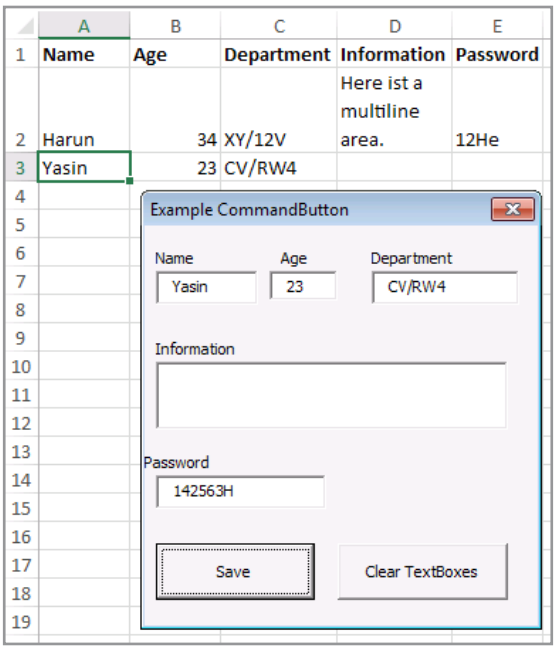


Figure 43: Result CommandButton

1.3.4 FRAME

A **Frame** can be recognized by the symbol  . The frame is a great way to group the controls into optically separate groups, or group radio buttons or check boxes.

If we want to create a frame over existing controls, we'll find that they're hidden by the inserted frame. To get them in the frame, we mark them, then cut and paste into the frame.

A frame in UserForm looks like this:

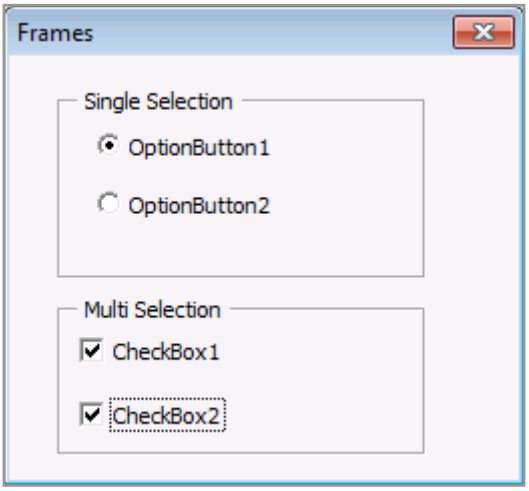


Figure 44: Example of Frame

1.3.5 OPTIONBUTTON

An **OptionButton** can be recognized by the symbol  . An option button is a single-choice. That is, with multiple radio buttons only one is selectable. If several option fields occur in a UserForm, only one of them can be selected. The radio buttons are similar to an enumeration with a round icon.

If we want to create multiple radio buttons on different criteria, we must group them together in different frames.

Then you can select multiple option fields of a UserForm, but only one option field per frame or per group.

Well, let us continue on to our example:

Our UserForm has a total of five radio buttons and a button to zero them, meaning set them back. We divide our UserForm into two groups. The upper group has the radio buttons “Field_1” to “Field_3” and the lower one has “Field_4” to “Field_5”. “Field_1” and “Field_2” are grouped under a GroupName = Group 1. “Field_3” remains alone. “Field_4” and “Field_5” are framed with a frame.

Grouping under a GroupName occurs in the UserForm_Initialize () procedure.
We can only select one of the grouped or framed radio buttons. The option field “Field_3” remains unaffected by this selection.

We set the option field “Field_2” in the VBA code with the .Value property to True, that is, we specify the selection.

As soon as an option field “field_1” to “field_5” is selected, the result in table 1 should be written from cell “A1” to “A5” as “FALSE” or “TRUE”. Field_1 and Field_2 and Field_4 and Field_5 cannot be the same because they belong to a group.

With the button “Reset option fields” all option fields should be set to not selected, i.e. to “FALSE”.

With the “Get jams” key, status results written in cells “D1” to “D5” are read in and selected accordingly. Before that we should edit the cells “D1: D5” by hand. Otherwise nothing can be read.

Our UserForm should look like this:

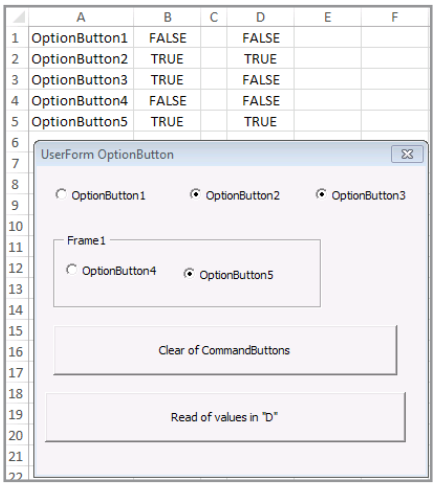


Figure 45: Example OptionButton

```
Option Explicit

Private Sub UserForm_Initialize()
  With OptionButton2
    .GroupName = "NewGroup"
    .Value = True
  End With
    OptionButton1.GroupName = "NewGroup"
End Sub

Private Sub Userform_Activate()
  Range("b1") = OptionButton1.Value
  Range("b2") = OptionButton2.Value
  Range("b3") = OptionButton3.Value
  Range("b4") = OptionButton4.Value
  Range("b5") = OptionButton5.Value
End Sub

Private Sub CommandButton1_Click()
  OptionButton1.Value = False
  OptionButton2.Value = False
  OptionButton3.Value = False
  OptionButton4.Value = False
  OptionButton5.Value = False
  Call Userform_Activate
End Sub

Private Sub CommandButton2_Click()
  OptionButton1.Value = Range("d1").Value
  OptionButton2.Value = Range("d2").Value
  OptionButton3.Value = Range("d3").Value
  OptionButton4.Value = Range("d4").Value
  OptionButton5.Value = Range("d5").Value
End Sub

Private Sub OptionButton1_Click()
  Range("b1") = OptionButton1.Value
  Call Userform_Activate
End Sub

Private Sub OptionButton2_Click()
  Range("b2") = OptionButton2.Value
  Call Userform_Activate
End Sub

Private Sub OptionButton3_Click()
  Range("b3") = OptionButton3.Value
  Call Userform_Activate
End Sub

Private Sub OptionButton4_Click()
  Range("b4") = OptionButton4.Value
  Call Userform_Activate
End Sub

Private Sub OptionButton5_Click()
  Range("b5") = OptionButton5.Value
  Call Userform_Activate
End Sub
```

1.3.6 CHECKBOX

A **CheckBox** can be recognized by the symbol  . This element is like the previous option element, but can be selected multiple times for several elements. With the value, the state of the element can be checked for activity with "**True**" or "**False**".

In this case we also want to enter the result in cells "B1" to "B4" or read them out of cells "D1" to "D4".

For the current example we bring colors into play. That is, depending on the selection, the color should be displayed in color in UserForm as well as in the result in the table. Use the keys to set all selections to "FALSE" and read in results from the sheet.

Before that we should edit the cells "D1: D4" by hand. Otherwise nothing can be read.

Our UserForm looks like this and the listing:

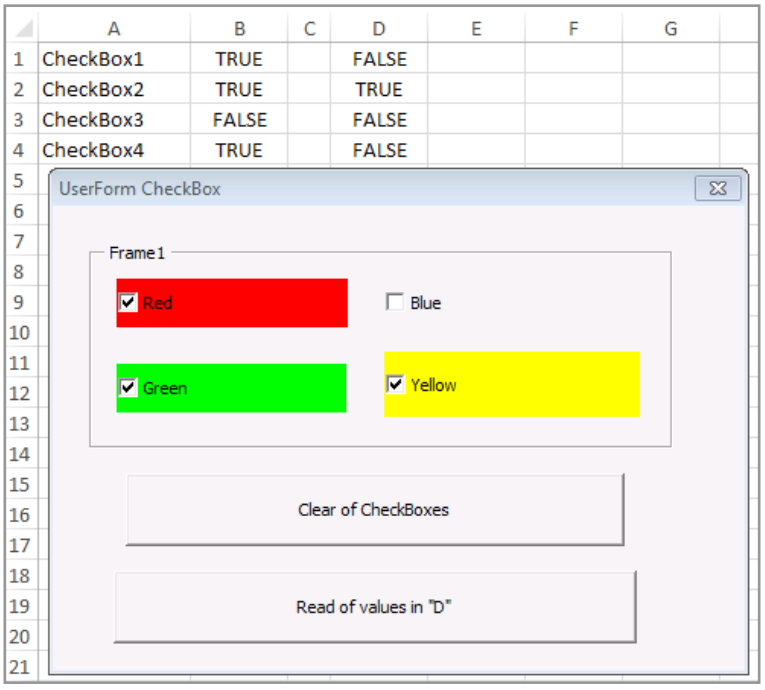


Figure 46: Example CheckBox

```
Private Sub CheckBox1_Click() 'red
  Select Case CheckBox1.Value
   Case True
      CheckBox1.BackColor = vbRed
   Case False
     CheckBox1.BackColor = &H8000000F
  End Select
Range("b1").Value = CheckBox1.Value
End Sub

Private Sub CheckBox2_Click() 'green
   Select Case CheckBox2.Value
   Case True
    CheckBox2.BackColor = RGB(0, 255, 0)
   Case False
   CheckBox2.BackColor = &H8000000F
  End Select
Range("b2").Value = CheckBox2.Value
End Sub
Private Sub CheckBox3_Click() 'blue
    Select Case CheckBox3.Value
     Case True
      CheckBox3.BackColor = RGB(0, 0, 255)
     Case False
      CheckBox3.BackColor = &H8000000F
   End Select
Range("b3").Value = CheckBox3.Value

Private Sub CheckBox4_Click() 'yellow
    Select Case CheckBox4.Value
     Case True
      CheckBox4.BackColor = RGB(255, 255, 0)
     Case False
      CheckBox4.BackColor = &H8000000F
    End Select
Range("b4").Value = CheckBox4.Value
End Sub

Private Sub CommandButton1_Click()
CheckBox1.Value = False
CheckBox2.Value = False
CheckBox3.Value = False
CheckBox4.Value = False
End Sub

Private Sub CommandButton2_Click()
CheckBox1.Value = Range("d1").Value
CheckBox2.Value = Range("d2").Value
CheckBox3.Value = Range("d3").Value
CheckBox4.Value = Range("d4").Value
End Sub
```

1.3.7 LISTBOX

A **ListBox** can be recognized by the symbol  .

A list box is nothing more than the contents of a table space. List box fillings from the range of a table is very convenient. For this purpose, either option field (single selection) or check box (multiple selection) can be displayed.

Well, let's look at different variants. For a fixed range, you can enter "A1: A20" as "Sheet1".

Fixed range:

```
.RowSource = "Sheet1!"A1:A20"
```

Dynamic range:

```
 'Renamed
 .RowSource = "New_Name"
 'Selected last cell in first row
 .RowSource = "Sheet1!"A1:" & Cells(Rows.Count, 1).End(xlUp).Row

 'The filling in UserForm_Initialize() Singleline:
 Private Sub UserForm_Initialize()
    Me.ListBox1.List = Worksheets("Sheets1").Range("A1").CurrentRegion.Columns(1).Value
 End Sub

 'The filling in UserForm_Initialize() Multiline:
Private Sub UserForm_Initialize()
  Me.ListBox1.List= _
  Worksheets("Sheet1").Range("A:B").CurrentRegion.Columns("A:B").Value
End Sub
```

Now we come to our first example with the option buttons. The function of our example is:

- .RowSource is realized by a renaming.
- The extension of the .RowSource area can be reached by pressing the button "Enter new name".
- Our selection is entered directly in cell "B1".
- Appear as desired by pressing the "Output selection" button.

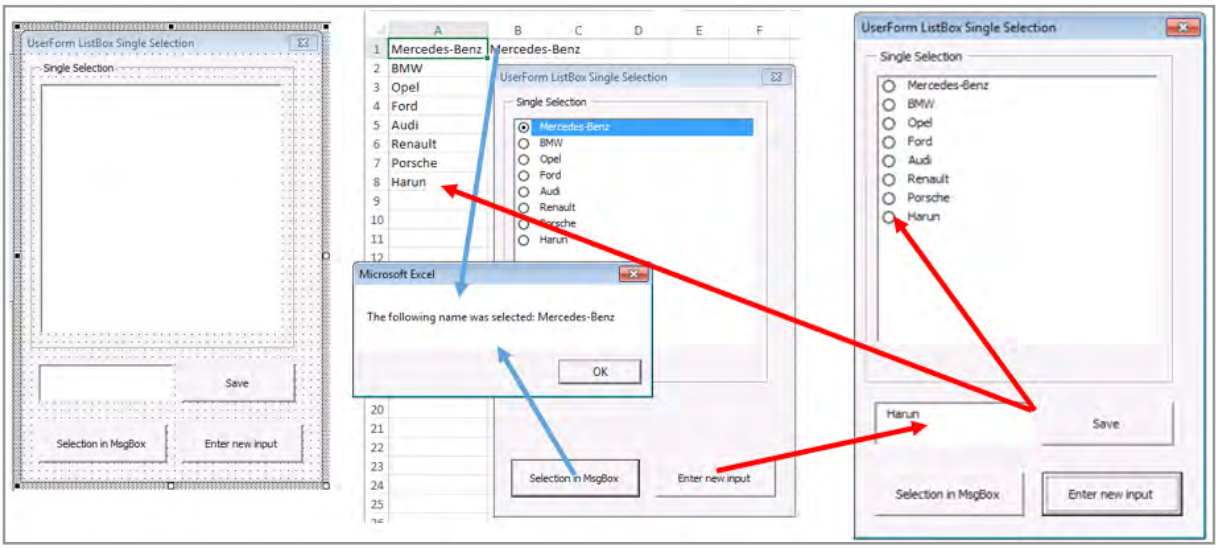


Figure 47: Example design and ListBox with single selection

In the following example we want to make the area to be listed dynamic.

```
Option Explicit
Private Sub CommandButton1_Click()
  If ListBox1.Value <> "" Then
    MsgBox "The following name was selected: " & ListBox1.Value
  End If
End Sub

Private Sub CommandButton2_Click() 'Enter new input
Dim i As Integer
For i = 0 To ListBox1.ListCount - 1
    ListBox1.Selected(i) = False
Next
TextBox1.Visible = True
CommandButton3.Visible = True
End Sub

Private Sub CommandButton3_Click()
  Dim Last_Cell As Integer
    'Insert in last cell
     Cells(Rows.Count, 1).End(xlUp).Offset(1, 0) = TextBox1.Value
   'Find last cell in row „A"
    Last_Cell = Cells(Rows.Count, 1).End(xlUp).Row
  'Renamed the the range
  ActiveWorkbook.Names.Add _
        Name:="New_Name", _
        RefersTo:=Range("A1:A" & Cells(Rows.Count, 1).End(xlUp).Row)
  'Refresh UserForm_Initialize
     Call UserForm_Initialize
End Sub

Private Sub ListBox1_Click() 'Selections in ListBox
  If ListBox1.Value <> "" Then
    Range("B1").Value = ListBox1.Value
  End If
End Sub

Private Sub UserForm_Initialize()
'Hidden
TextBox1.Visible = False
CommandButton2.Visible = False
'Multiselection
  ActiveWorkbook.Names.Add _
        Name:="New_Range", _
        RefersTo:=Range("A1:A" & Cells(Rows.Count, 1).End(xlUp).Row)
'Create ListenBox
  With ListBox1
    .RowSource = "New_Name"
    .MultiSelect = fmMultiSelectSingle
    .ListStyle = fmListStyleOption
  End With
End Sub
```

The presentation with a circle in front of each listing looks aesthetically pleasing. We have the opportunity to omit this circle symbol. To do this we should change the key to .ListStyle = fmListStylePlain in the VBA listing "UserForm_Initialize".

```
Private Sub UserForm_Initialize()
'Hidden
TextBox1.Visible = False
CommandButton2.Visible = False
'Create ListBox
  With ListBox1
    .RowSource = "New_Name"
    .MultiSelect = fmMultiSelectSingle
    .ListStyle = fmListStylePlain
  End With
End Sub
```

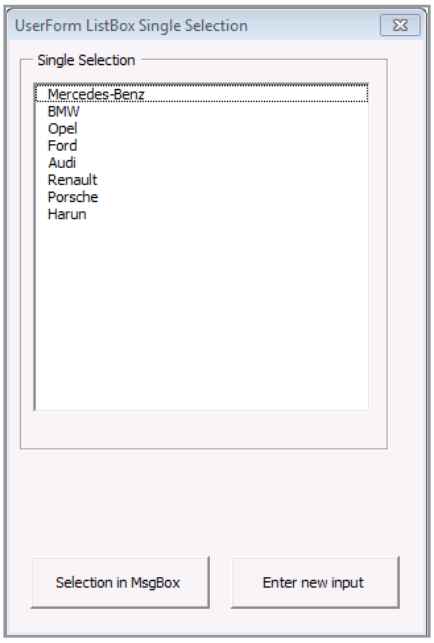


Figure 48: Example ListBox single selection without circle

Before we jump to the next example, a few notes on multiple selection:
"0" is used to mark the first entry as VBA starts counting from zero.

```
ListBox1.ListIndex=0
```

To reset all markers we use a For..Next loop as follows:

```
For i=0 To ListBox1.ListCount-1
        ListBox1.Selected(i)=False
Next i
```

How should our current example work?
Column "A" lists several car brands. This area is defined as "New Area" and shown in the list box. Selected automobile brands are entered in column "B" with the "Output selection" button and shown in the "Selection window".
With the key "enter mark" an input field appears with "Save" - key. Before that, selections are reset in the background when the records have been selected. Then you can enter further data records.

The previous example with multiple selection.

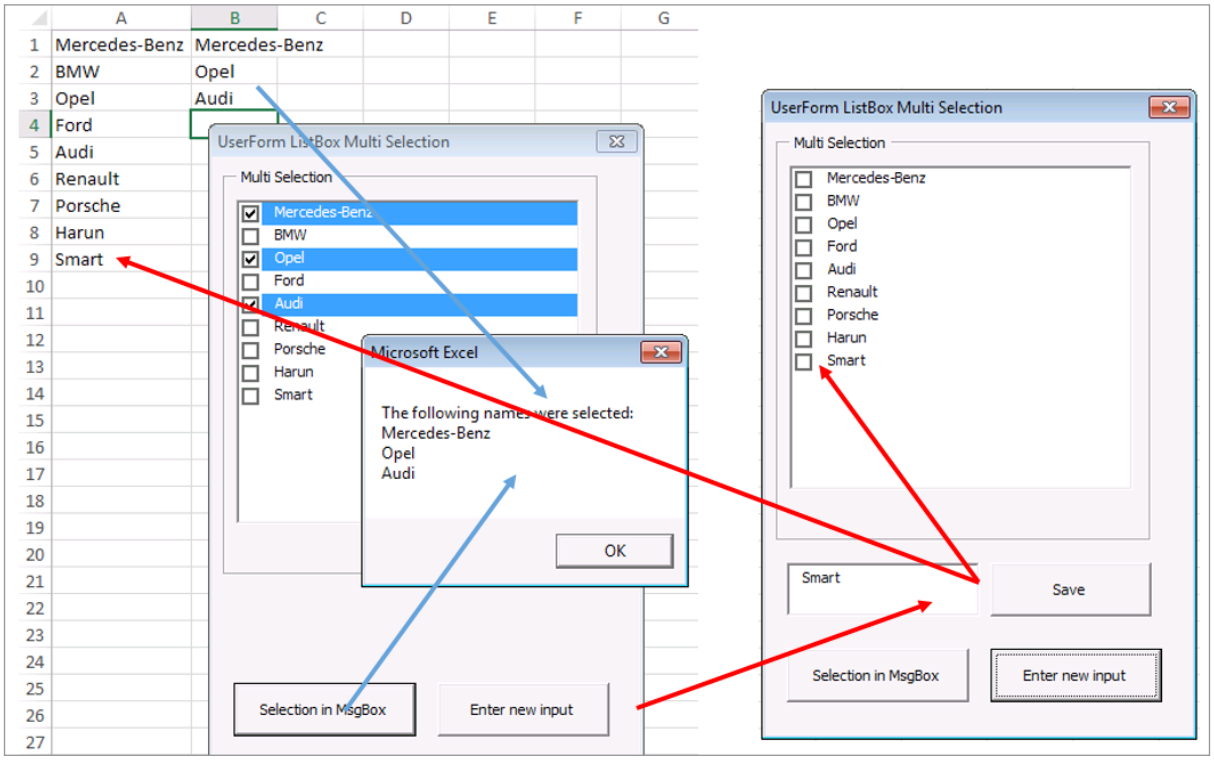


Figure 49: Example ListBox with multi selection

The VBA code looks like the single-selection. The difference from the previous example is as shown below:

```
Private Sub UserForm_Initialize()
'Hidden
TextBox1.Visible = False
CommandButton3.Visible = False
'Create ListBox
  With ListBox1
    .RowSource = "New_Name"
    .MultiSelect = fmMultiSelectMulti
    .ListStyle = fmListStyleOption
  End With
End Sub

Private Sub CommandButton1_Click()
Dim i, y As Integer
Dim strName As String
Range("b1").Select
  For i = 0 To ListBox1.ListCount - 1
    If ListBox1.Selected(i) = True Then
      ActiveCell.Offset(0, 0).Value = ListBox1.List(i)
      ActiveCell.Offset(1, 0).Select
      strName = strName & ListBox1.List(i) & vbCrLf
   End If
 Next i
If strName <> "" Then
    MsgBox "The following names were selected:" & vbCrLf & strName
End If
End Sub
```

Again, a square before each listing looks very good. If we want to omit this square icon, simply change the key to **.ListStyle = fmListStylePlain** in the VBA listing "**UserForm_Initialize**".

So far, we have specified our list box fill with ListBox1.RowSource = "Range". Now we will see an example with AddItem:

In the current example we have four entries with AddItem. Entries selected by single selection are entered in cell "A1". First we see our UserForm, then the listing:

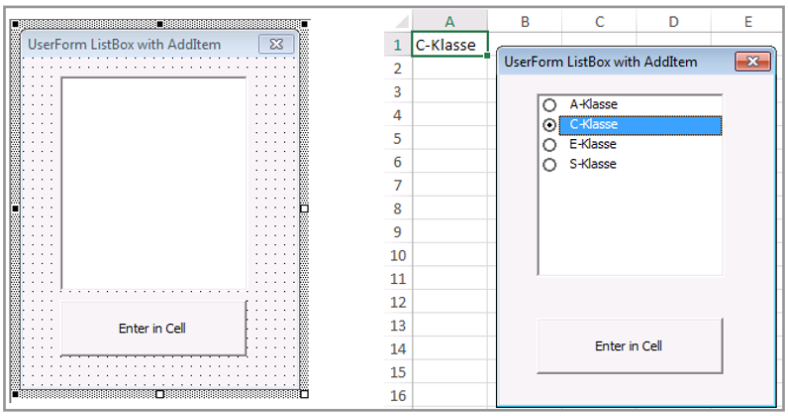


Figure 50: Filling ListBox (single selection) with AddItem

The listing:

```
Private Sub UserForm_Initialize() 'Single selection
    With UserForm1.ListBox1
        .AddItem "A-Klasse"
        .AddItem "C-Klasse"
        .AddItem "E-Klasse"
        .AddItem "S-Klasse"
    End With
 Worksheets("Sheet1").Activate
 ListBox1.ListStyle = fmListStyleOption
End Sub

Private Sub CommandButton1_Click() 'Single selection
 Range("A1").Value = UserForm1.ListBox1.Value
End Sub
```

With multiple selection should look like this:

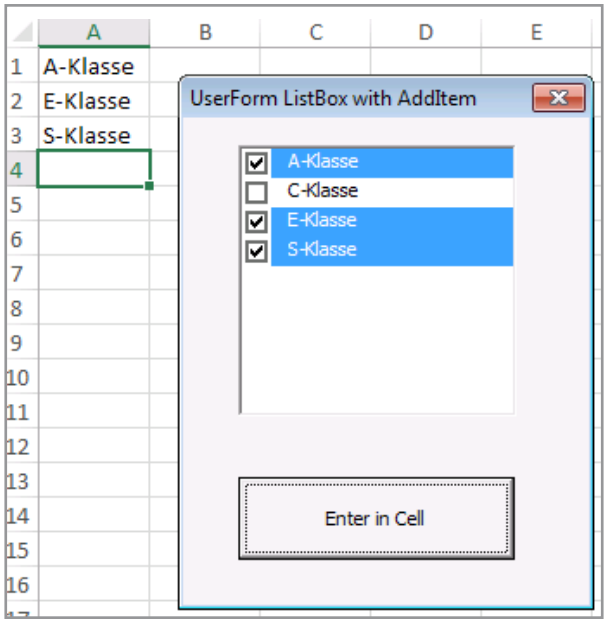


Figure 51: Filling ListBox (multi selection) with AddItem

The listing looks like this:

```
Private Sub UserForm_Initialize() 'Multiselection
    With UserForm1.ListBox1
        .AddItem "A-Klasse"
        .AddItem "C-Klasse"
        .AddItem "E-Klasse"
        .AddItem "S-Klasse"
    End With
Worksheets("Sheet1").Activate
With ListBox1
    .ListStyle = fmListStyleOption
    .MultiSelect = fmMultiSelectMulti
End With
End Sub

Private Sub CommandButton1_Click() 'Multiselection
Range("A1").Select
For i = 0 To ListBox1.ListCount - 1
    If ListBox1.Selected(i) = True Then
      ActiveCell.Offset(0, 0).Value = ListBox1.List(i)
      ActiveCell.Offset(1, 0).Select
      strName = strName & ListBox1.List(i) & vbCrLf
   End If
 Next i
End Sub
```

1.3.8 LISTBOX WITH MULTI COLUMN

So far, we have covered single-column list boxes. We also have the option of presenting multi-column list boxes with a headline.

Our design looks like this:

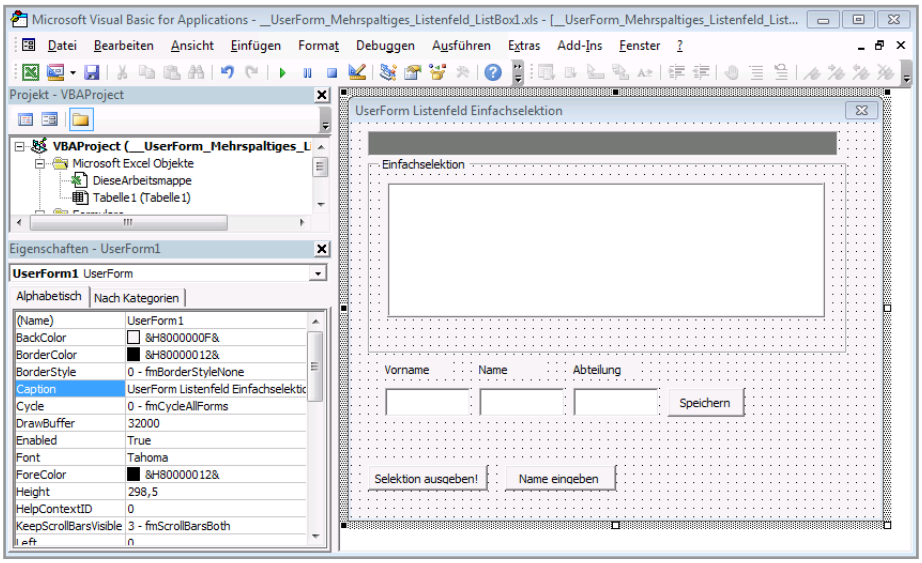


Figure 52: UserForm design to ListBox with multi column

This is how a finished construct and matching table with entries should look like:

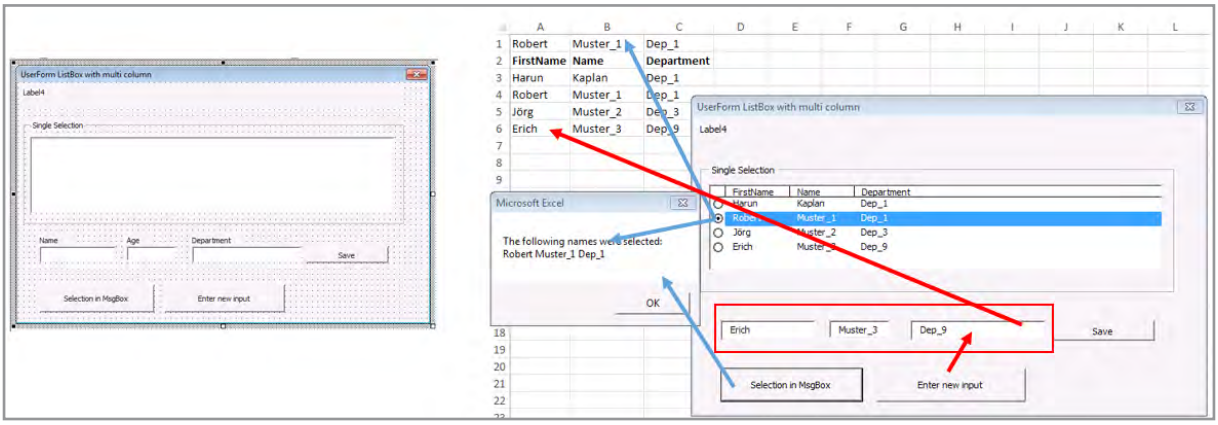


Figure 53: Example ListBox multi column

To represent a multicolumn Listbox, we will write the following properties in the VBA code:

- **.ColumnsCount** indicates the number of columns,
- **.RowSource** indicates the source without a heading,
- **.ColumnHead** specifies the heading,
- **.ColumnWidths** specifies the column width,
- **.ListStyle** indicates the appearance of the enumeration.

Then, the VBA code for this:

```
Private Sub UserForm_Initialize()
Dim x As Integer
With ListBox1
    .ColumnCount = 3
    .RowSource = "New_Name"
    .ColumnHeads = True
    .ColumnWidths = "2cm; 2cm; 2cm"
    .ListStyle = fmListStyleOption
End With
 Label1.Visible = False
 Label2.Visible = False
 Label3.Visible = False
 TextBox1.Visible = False
 TextBox2.Visible = False
 TextBox3.Visible = False
 CommandButton3.Visible = False
End Sub

Private Sub CommandButton1_Click()
Dim a, b, c, d, strName As String
   With Me.ListBox1
      a = .List(.ListIndex, 0)
      b = .List(.ListIndex, 1)
      c = .List(.ListIndex, 2)
   End With
strName = a & " " & b & " " & c
       If strName <> "" Then
          MsgBox "The following names were selected: " & vbCrLf & strName
       End If
End Sub

Private Sub CommandButton3_Click()
  Cells(Rows.Count, 1).End(xlUp).Offset(1, 0) = TextBox1.Value
  Cells(Rows.Count, 2).End(xlUp).Offset(1, 0) = TextBox2.Value
  Cells(Rows.Count, 3).End(xlUp).Offset(1, 0) = TextBox3.Value
  Last_cell = Cells(Rows.Count, 1).End(xlUp).Row
  ActiveWorkbook.Names.Add _
       Name:="New_Name", _
       RefersTo:=Range("A3:C" & last_cell)
Call UserForm_Initialize
End Sub

Private Sub CommandButton2_Click()
 'Clearing
 Label1.Caption = ""
 Label2.Caption = ""
 Label3.Caption = ""
 TextBox1.Value = ""
 TextBox2.Value = ""
 TextBox3.Value = ""
 'Make invisible elements visible
 Label1.Visible = True
```

```
 Label2.Visible = True
 Label3.Visible = True
 '
 TextBox1.Visible = True
 TextBox2.Visible = True
 TextBox3.Visible = True
 '
 CommandButton3.Visible = True
End Sub

Private Sub ListBox1_Click()
Dim CellA1, CellB1, CellC1 As String
With ListBox1
   CellA1 = .List(.ListIndex, 0)
   CellB1 = .List(.ListIndex, 1)
   CellC1 = .List(.ListIndex, 2)
End With
   Range("a1").Value = CellA1
   Range("b1").Value = CellB1
   Range("c1").Value = CellC1
End Sub

Private Sub ListBox1_Change()
Dim Value1, Value2, Value3 As String
With ListBox1
   Value1 = .List(.ListIndex, 0) 'Name
   Value2 = .List(.ListIndex, 1) 'First name
   Value3 = .List(.ListIndex, 2) 'Department
End With
Label4.Caption = Value1 & " - " & Value2 & " - " & Value3
End Sub
```

Similar to the previous chapter where we used AddItem to filled a single-column ListBox, we can also fill a multi-column ListBox.

We first determine what our list box should look like. Our details are:

- Number of columns: ListBox1.ColumnCount = 4;
- If uniform column width is desired: ListBox1.BoundClumn = 1;
- Desired column width: ListBox1.ColumnWidths = “2.5cm; 4cm; 2mm; 2.5mm”;
- Width of the ListBox window: ListBox1.ColumnCount = 200;
- If any data is included, empty with ListBox1.Clear.

Now we can fill our ListBox with the contents of the table using a For..Next loop.

The UserForm looks like this:

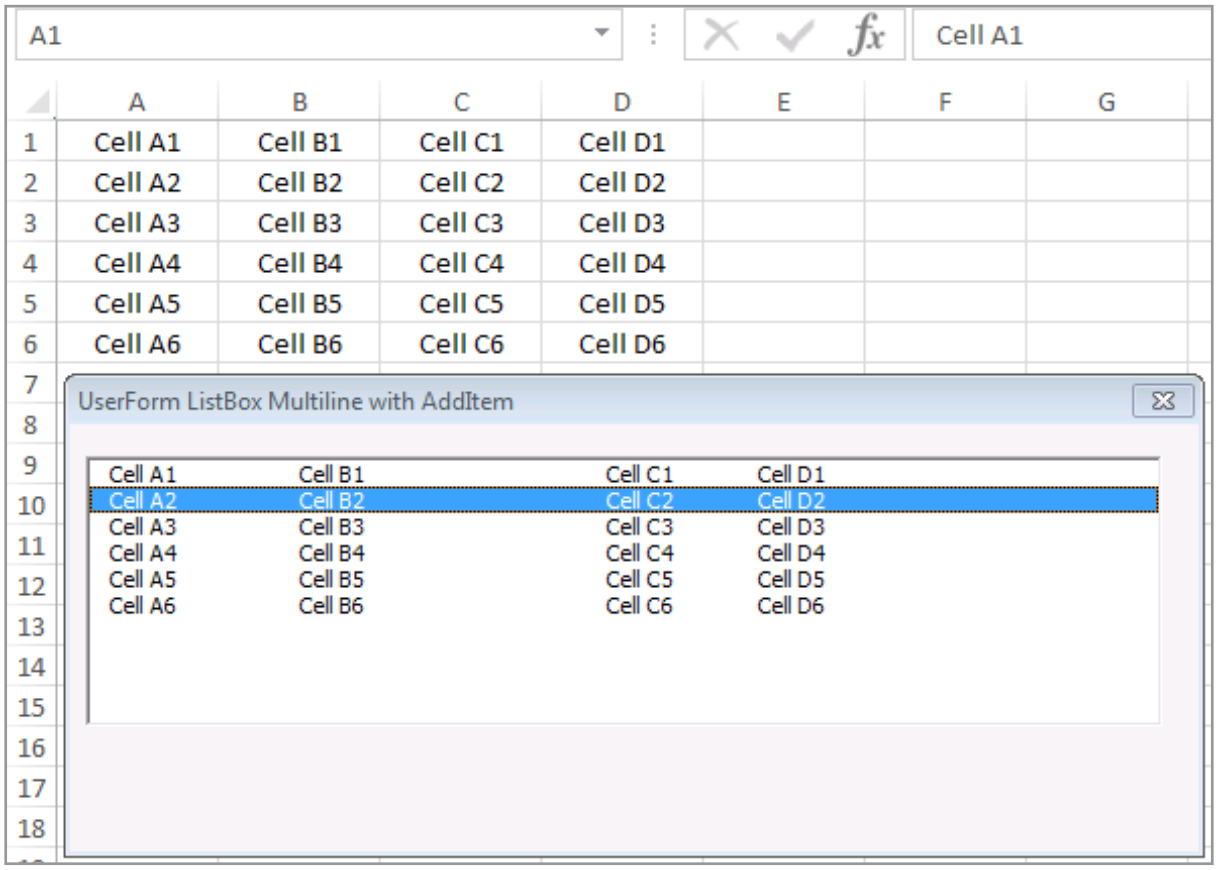


Figure 54: Example filling a ListBox with AddItem

The listing looks like this:

```
Private Sub UserForm_Initialize()
  Dim i As Integer
  With Me.ListBox1
    .ColumnCount = 4
    .ColumnWidths = "2,5cm;4cm;2cm;3cm"
    .Width = 200
    .Clear
    For i = 1 To 6
      .AddItem Cells(i, 1).Value
      .List(.ListCount - 1, 1) = Cells(i, 2).Value
      .List(.ListCount - 1, 2) = Cells(i, 3).Value
      .List(.ListCount - 1, 3) = Cells(i, 4).Value
    Next
  End With
End Sub
```

1.3.9 DROPDOWN WITH COMBOBOX

A **Dropdown with ComboBox** can be recognized by the symbol  . This control is like a drop-down list box with an arrow key on the right side. If we open this dropdown field, we will see one or more lines of data depending on the setting

```
'The filling can be directly in UserForm_Initialize() automatized (two columns here):
Private Sub UserForm_Initialize()
        Me.ComboBox1.List = _
         Worksheets("Sheet1").Range("A:B").CurrentRegion.Columns("A:B").Value
End Sub
```

For this we will use the example of multi-line list box and complete it with a combo box.

We have three options here

1. use the list box to select our selection. As a result, the combo box is filled with the respective first name and lower label field with the content “First name & name & department”.
2. The selection is selected via the combo box and selected in the multi-line list box. The label field is filled accordingly. The combo box is displayed with all of the columns and the header label.
3. Getting the information through ComboBox

Our UserForm should look like the design below.

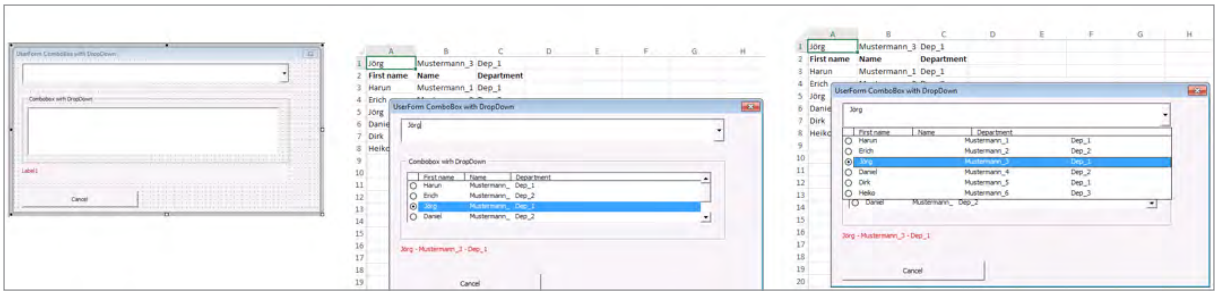


Figure 55: Example ComboBox with Dropdown

Here is our Listing :

```
Option Explicit

Private Sub ComboBox1_Change()
Dim intSelectIndex As Integer
  intSelectIndex = ComboBox1.ListIndex
  Range("A1").Value = ComboBox1.Value
End Sub

Private Sub CommandButton1_Click()
Unload Me
End Sub

Private Sub ListBox1_Click()
  With ListBox1
    Range("a1").Value = .List(.ListIndex, 0)
    Range("b1").Value = .List(.ListIndex, 1)
    Range("c1").Value = .List(.ListIndex, 2)
  End With
End Sub

Private Sub ListBox1_Change()
Dim strSelection1, strSelection2, strSelection3 As String
  With ListBox1

    strSelection1 = .List(.ListIndex, 0)
    strSelection2 = .List(.ListIndex, 1)
    strSelection3 = .List(.ListIndex, 2)
  End With
  ComboBox1.Value = ListBox1.List(ListBox1.ListIndex, 0) ' Pass value to combobox
  Label1.Caption = strSelection1 & " - " & strSelection2 & " - " & strSelection3
End Sub

Private Sub UserForm_Initialize()
  Dim Last_Cell As Integer
  Last_Cell = Cells(Rows.Count, 1).End(xlUp).Row

  ActiveWorkbook.Names.Add _
       Name:="New_Name", _
       RefersTo:=Range("A3:C" & Last_Cell)

With ListBox1
    .ColumnCount = 3
    .RowSource = "New_Name"
    .ColumnHeads = True
    .ColumnWidths = "2cm; 2cm; 2cm"
    .ListStyle = fmListStyleOption
    .Selected(2) = True ' third entry preselected
  End With

  With ComboBox1
    .ColumnCount = 3
    .RowSource = "New_Name"
    .ColumnHeads = True
    .ListStyle = fmListStyleOption
  End With
End Sub
```

Allow only list entries

If we just want to allow list entries, the Style property of the ComboBox will change to **2-frmStyleDropDownList**.
The change to the Style property looks like this:

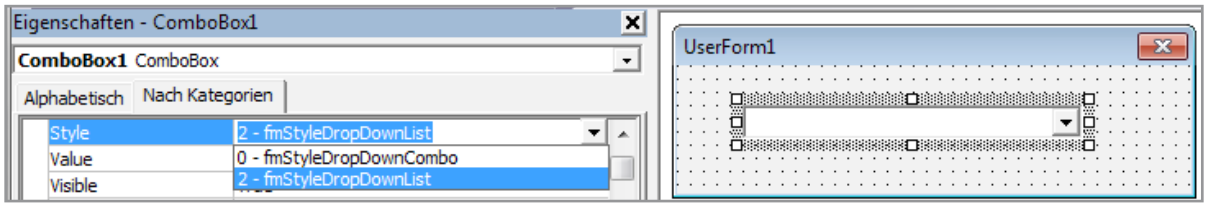


Figure 56: Setting of Style!

As soon as we enter a letter of an entry, the first possible entry appears in blue.
Now this is the result:

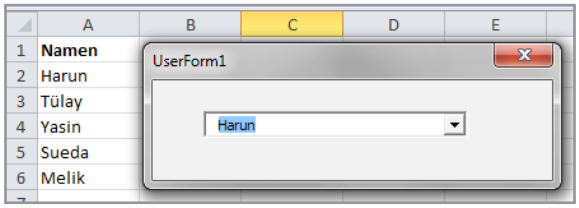


Figure 57: Style at work

1.3.10 SPINBUTTON

A **SpinButton** can be recognized by the symbol  .
We have certainly searched for a radio station several times on a car radio by using a rotatable button. That's exactly how it works here. We can either turn up / down or right / left, click here.

Our current example has two spin fields. The right spin moves the cursor up or down. The lower rotating field, on the other hand, moves the cursor to the left or to the right. We will print the address of the cursor and the content of the current cell in Label. At the same time, we allow the content to be written to cell "D1".

Our UserForn should look like next picture:

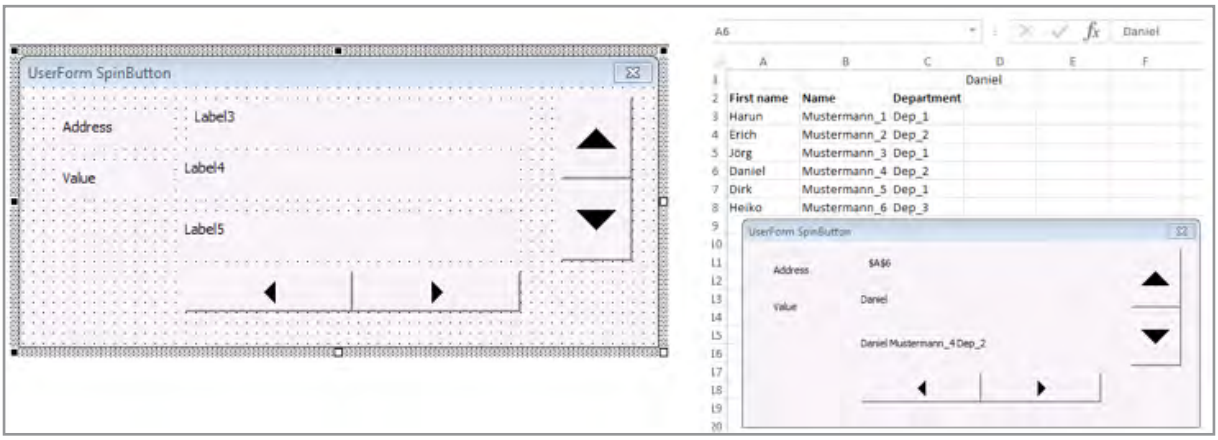


Figure 58: Example SpinButton

When moving the cursor, if it reaches an edge no error message should be displayed. The cursor should move only when it can actually move in the desired direction. Therefore, after the click we will search the current line, "intRow" or the current column, "intColumn".

We will then write a procedure in each direction because this element practically consists of two different elements.

```
Private Sub SpinButton2_SpinDown() 'downward
Dim intRow As Integer
  intRow = ActiveCell.Row
  If intRow <= Cells(Rows.Count, 1).End(xlUp).Row Then
     ActiveCell.Offset(1, 0).Select
     Call Info
  End If
End Sub

Private Sub SpinButton2_SpinUp() 'up
Dim intRow As Integer
  intRow = ActiveCell.Row
  If intRow > 1 Then
     ActiveCell.Offset(-1, 0).Select
     Call Info
  End If
End Sub

Private Sub SpinButton1_SpinDown() 'to left
Dim intColumn, intRow As Integer
  intColumn = ActiveCell.Column
  If intSpalte > 1 Then
     ActiveCell.Offset(0, -1).Select
     Call Info

  End If
End Sub
```

```
Private Sub SpinButton1_SpinUp() 'to right
Dim intColumn, intRow As Integer
  intColumn = ActiveCell.Column
  If intSpalte < Cells(2, Columns.Count).End(xlToLeft).Column Then
      ActiveCell.Offset(0, 1).Select
      Call Info
  End If
End Sub

Sub Info()
Dim Active_Row As Integer
Dim First_Name, Name, Department As String
      Label4.Caption = ActiveCell.Value
      Label3.Caption = ActiveCell.Address
      Range("D1").Value = Label4.Caption
      First_Name = Cells(ActiveCell.Row, 1).Value
      Name = Cells(ActiveCell.Row, 2).Value
      Department = Cells(ActiveCell.Row, 3).Value
      Label5.Caption = First_Name & " " & Name & " " & Department
End Sub
```

1.3.11 SCROLLBAR

A **ScrollBar** can be recognized by the symbol  . This control is similar to the spinner element from earlier. This element has a horizontal slide. The maximum and minimum values of the horizontal slider can also be entered in the Properties window at **Max** and **Min**.

Well, let's first look at the design and the result:

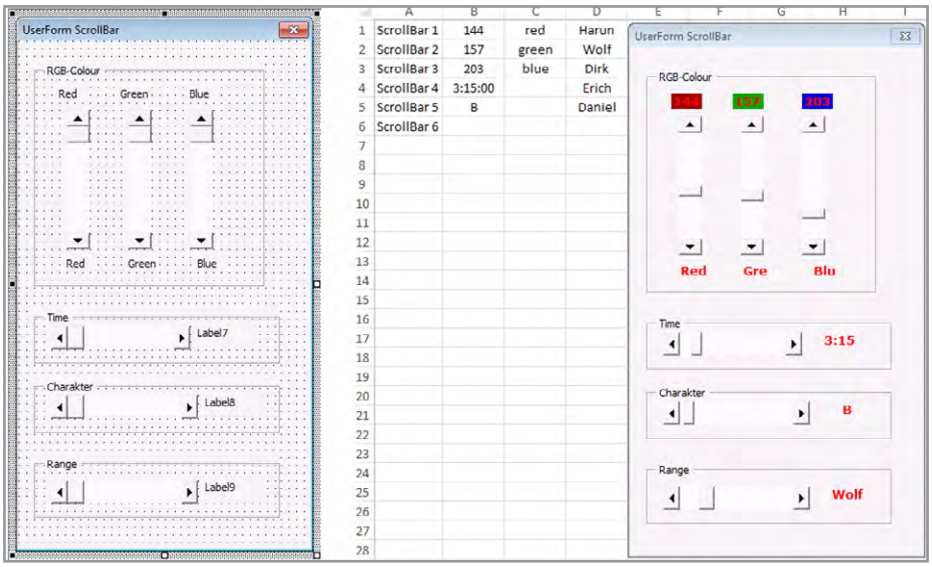


Figure 59: Example ScrollBar

Here is our Listing:

```
Private Sub ScrollBar1_Change() 'red
  With ScrollBar1
    .Min = 0
    .Max = 255
  End With
  With Label1
   .Caption = ScrollBar1.Value
   .BackColor = RGB(ScrollBar1.Value, 0, 0)
  End With
  With Range("B1")
   .Value = ScrollBar1.Value
  End With
End Sub

Private Sub ScrollBar2_Change() 'green
  With ScrollBar2
    .Min = 0
    .Max = 255
  End With
  With Label2
   .Caption = ScrollBar2.Value
   .BackColor = RGB(0, ScrollBar1.Value, 0)
  End With
```

```
  With Range("B2")
   .Value = ScrollBar2.Value
  End With
End Sub

Private Sub ScrollBar3_Change() 'blue
  With ScrollBar3
    .Min = 0
    .Max = 255
  End With
  With Label3
   .Caption = ScrollBar3.Value
   .BackColor = RGB(0, 0, ScrollBar1.Value)
  End With
  With Range("B3")
   .Value = ScrollBar3.Value
  End With
End Sub

Private Sub ScrollBar4_Change() 'Time
Dim Minutes As String
Dim i As Integer

  With ScrollBar4
    .Min = 0
    .Max = 24
  End With
i = Int(15 * Rnd)

  Select Case i
   Case 0, 4, 8, 12
    Minutes = ":00"
    i = i + 1
  Case 1, 5, 9, 13
   Minutes = ":15"
   i = i + 1
 Case 2, 6, 10, 14
   Minutes = ":30"
   i = i + 1
 Case 3, 7, 11, 15
   Minutes = ":45"
  i = 1 + 1
 End Select

  If ScrollBar4.Value = 0 Or ScrollBar4.Value = 24 Then
   Minutes = ":00"
  End If

  With Label7
   .Caption = ScrollBar4.Value & Minutes
  End With

  With Range("B4")
    .Value = ScrollBar4.Value & Minutes
  End With
End Sub
```

```
Private Sub ScrollBar5_Change() 'Time
Dim i As Integer
  With ScrollBar5
    .Min = 65
    .Max = 90
  End With
  With Label5
    .Caption = Chr(ScrollBar5.Value)
  End With
  With Range("B5")
    .Value = Chr(ScrollBar5.Value)
  End With
End Sub

Private Sub ScrollBar6_Change() 'Range
Dim intCount As Integer
intCount = Cells(Rows.Count, 1).End(xlUp).Row
With ScrollBar6
  .Min = 1
  .Max = intCount
End With

With Label9
 .Caption = Cells(ScrollBar6.Value, 4).Value
End With

End Sub

Private Sub UserForm_Initialize()
 Dim i As Integer
 For i = 1 To 9
   With Me.Controls("Label" & i) 'Label1
      .TextAlign = fmTextAlignCenter
     .Font.Bold = True
     .Font.Size = 10
     .Font.Name = "Verdana"
    .ForeColor = vbRed
  End With
 Next
End Sub
```

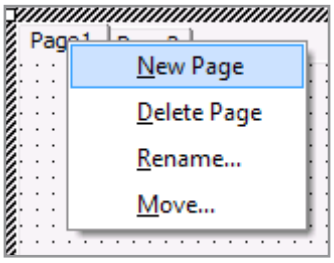
1.3.12 MULTIPAGE

A **MultiPage** can be recognized by the symbol  . If we look closely, we will find that individual pages are similar to a multi-page, such as a frame or even a UserForm.

Each multi-page can have any number of different elements

TextBox; Listbox, OptionButton, and so on.
Each page is individually addressable
Page1; Page2, and so on.

The creation of the control elements make to two pages, “Page1” and “Page2”, automatically available. We can customize automatic naming as follows:
In the property of the respective multi-page or
right-click on the page label
open a context menu with the following options:



- “**New Page**“ attaches a new page,
- “**Delete Page**“ deletes selected pages,
- “**Rename**“ changes the name of „PageX“,
- “**Move**“ shifts the selected page.

In our example, we are using two multi-pages as “Information_1” and “Information_2”; a spin button to move the cursor in the data area; two buttons to insert new record once, then save inserted record. Here we can also change existing record and save.

Here is how our design should look:

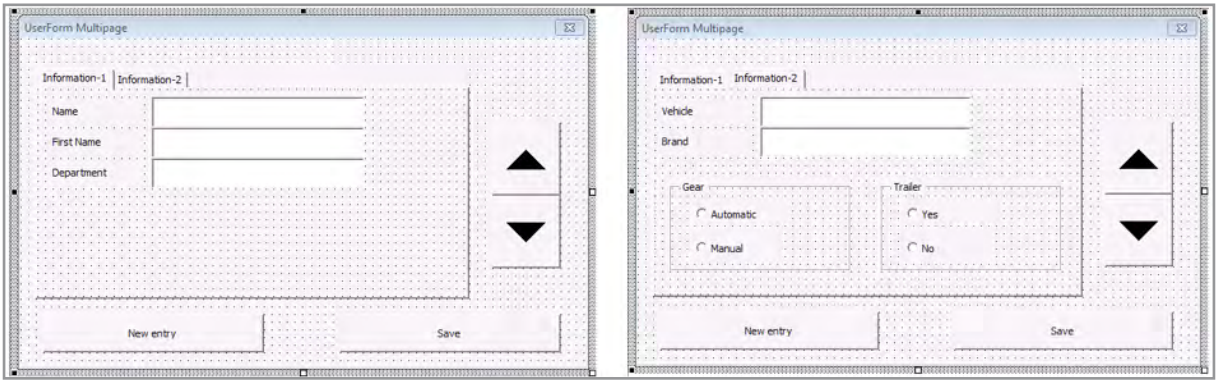


Figure 60: Design of MultiPage

If we want to see “Information 1” at the top after each startup, write MultiPage1.Value = 0. If we want “Information 2”, then we write “MultiPage.Value = 1” because it starts counting at “0”.

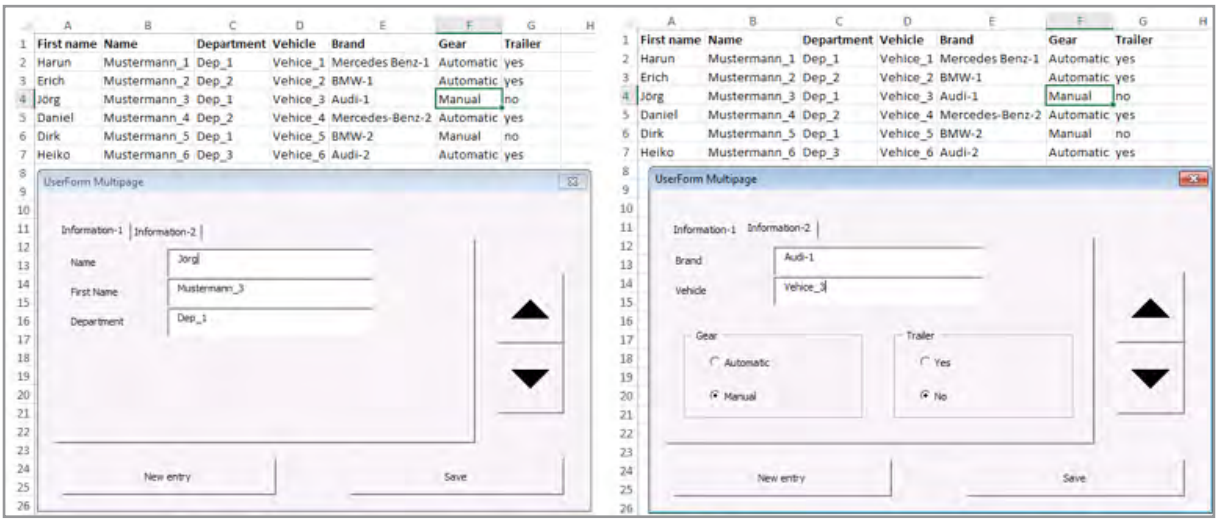


Figure 61: Counted from Page1 and Page2

The Listing:

```
Private Sub UserForm_Initialize()
   UserForm1.MultiPage1.Value = 0
   Call Info
End Sub

Private Sub CommandButton1_Click()
  Last_Cell = Worksheets("Sheet1").Cells(Rows.Count, 1).End(xlUp).Row
  Cells(Last_Cell + 1, 1).Select
  Call Info
End Sub

Private Sub CommandButton2_Click()
    Cells(ActiveCell.Row, 1).Value = TextBox1.Value
    Cells(ActiveCell.Row, 2).Value = TextBox2.Value
    Cells(ActiveCell.Row, 3).Value = TextBox3.Value
    Cells(ActiveCell.Row, 4).Value = TextBox4.Value
    Cells(ActiveCell.Row, 5).Value = TextBox5.Value
    If OptionButton1.Value = True Then
        Cells(ActiveCell.Row, 6).Value = "Automatik"
    Else
        Cells(ActiveCell.Row, 6).Value = "Manual"
    End If
    If OptionButton1.Value = False And OptionButton2.Value = False Then Cells(ActiveCell.Row,
6).Value = ""

    If OptionButton3.Value = True Then
        Cells(ActiveCell.Row, 7).Value = "yes"
    Else
        Cells(ActiveCell.Row, 7).Value = "no"
    End If
    If OptionButton3.Value = False And OptionButton4.Value = False Then Cells(ActiveCell.Row,
7).Value = ""
End Sub

Private Sub SpinButton1_SpinDown()
Dim intColumn, intRow As Integer
  intRow = ActiveCell.Row
  If intRow <= Cells(Rows.Count, 1).End(xlUp).Row Then
```

```
      ActiveCell.Offset(1, 0).Select
      Call Info
  End If
End Sub

Private Sub SpinButton1_SpinUp()
Dim intColumn, intRow As Integer
  intRow = ActiveCell.Row
  If intRow > 1 Then
      ActiveCell.Offset(-1, 0).Select
      Call Info
  End If
End Sub

Sub Info()
Dim aktive_row As Integer
Dim First_Name, Name, Department As String
    First_Name = Cells(ActiveCell.Row, 1).Value
    Name = Cells(ActiveCell.Row, 2).Value
    Department = Cells(ActiveCell.Row, 3).Value
    Vehicle = Cells(ActiveCell.Row, 4).Value
    Brand = Cells(ActiveCell.Row, 5).Value
    Gear = Cells(ActiveCell.Row, 6).Value
    Trailer = Cells(ActiveCell.Row, 7).Value
    '
    TextBox1.Value = First_Name
    TextBox2.Value = Name
    TextBox3.Value = Department
    TextBox4.Value = Vehicle
    TextBox5.Value = Brand
    If Gear = "Automatic" Then
        OptionButton1.Value = True
        OptionButton2.Value = False
    ElseIf Gear = "Manual" Then
        OptionButton1.Value = False
        OptionButton2.Value = True
    Else
        OptionButton1.Value = False
        OptionButton2.Value = False
    End If
    If Trailer = "yes" Then
        OptionButton3.Value = True
        OptionButton4.Value = False
    ElseIf Trailer = "no" Then
        OptionButton3.Value = False
        OptionButton4.Value = True
    Else
        OptionButton3.Value = False
        OptionButton4.Value = False
    End If
End Sub
```

1.3.13 TABSTRIP

A **TabStrip** can be recognized by the symbol  . This element looks very similar to the Multipage. The differences from multi-page are:

There is only one tab “TabStrip1”
All controls belong to TabStrip1.

Here's a list of some TabStrip statements. We did not use them here, but they may be useful later:
Number of tabs: TapStrip1.Tabs.Count
Current Tab: TabStrip1.Value Height and width of tabs: TabStrip1.TabFixedHight = 25; TabStrip1.TabFixedWidth = 25

In our example, we are a Mercedes-Benz dealership. We have a table listing all vehicles in the yard. They are subdivided into "sedan, coupe", "convertible" and "offroad / SUV"
We would like to make this visible using a ListBox. Then we enter the selected vehicle in cell "F1" by "Enter selection in the table!".

This is what our design and finished UserForm looks like:

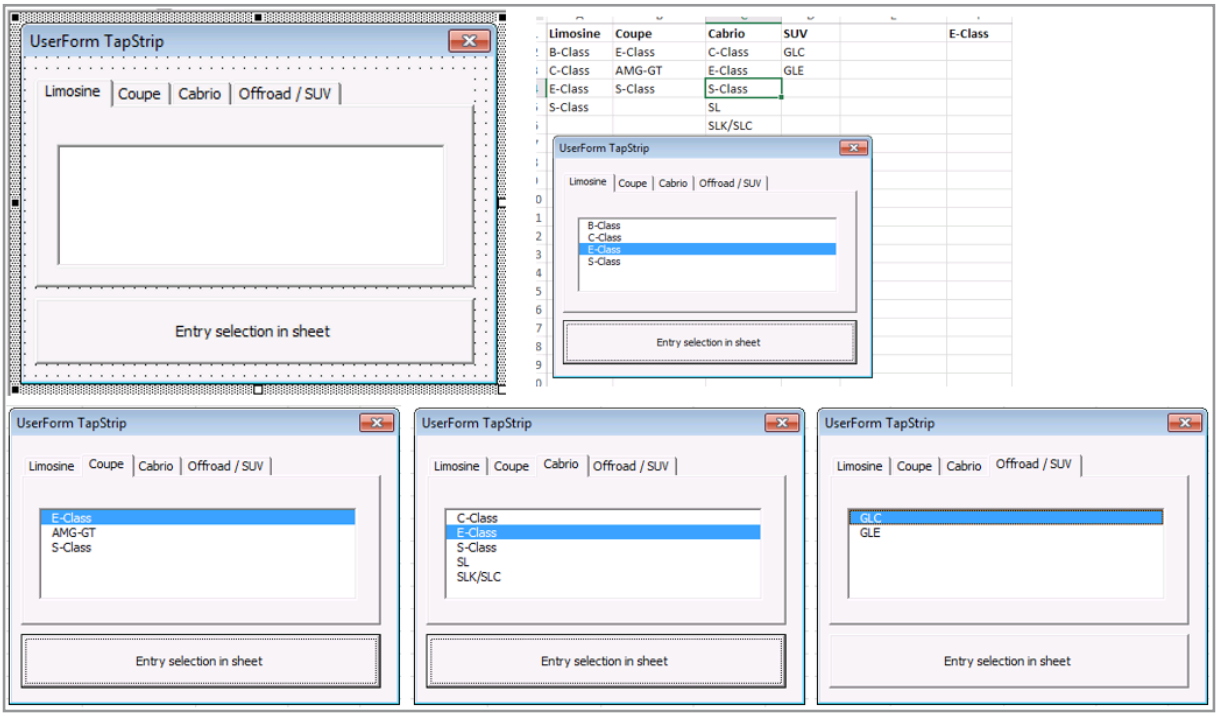


Figure 62: Example for TabStrip

The VBA-Listing:

```
Private Sub Userform_Initialize()
TabStrip1.Value = 0
Call TabStrip1_Change
End Sub

Private Sub TabStrip1_Change()
Dim StrRow As String
Dim intEnd As Integer
Worksheets("Vehicle").Select
Select Case TabStrip1.Value
    Case Is = 0 'Limousine
```

```
        intRow = "A"
        intEnd = Cells(Rows.Count, 1).End(xlUp).Row
        GoTo continue
    Case Is = 1 'Coupe
        intRow = "B"
        intEnd = Cells(Rows.Count, 2).End(xlUp).Row
        GoTo continue
    Case Is = 2 'Cabrio
        intRow = "C"
        intEnd = Cells(Rows.Count, 3).End(xlUp).Row
        GoTo continue
    Case Is = 3 'Offroad/SUV
        intRow = "D"
        intEnd = Cells(Rows.Count, 4).End(xlUp).Row
        GoTo continue
End Select
Exit Sub
continue:
  ActiveWorkbook.Names.Add _
        Name:="New_Range", _
        RefersTo:=Range(intRow & "2:" & intRow & intEnd)
'Listenquelle
  ListBox1.RowSource = "New_Range"
End Sub

Private Sub CommandButton1_Click()
Worksheets("Vehicle").Activate
ActiveSheet.Cells(1, 6).Value = Me.ListBox1.Value
End Sub
```

1.3.14 REFEDIT

A **RefEdit** (Reference Selection Edit) can be recognized by the symbol  . I personally have not used this item yet. Therefore, we will look at a possible use. In our example, the selected area is highlighted and the values are summed up. The result is entered in cell "A1".

The selected range is stored as a Value property and is represented as a string with the cell reference, for example, "Sheet! $ A $ 1: $ F $ 25".

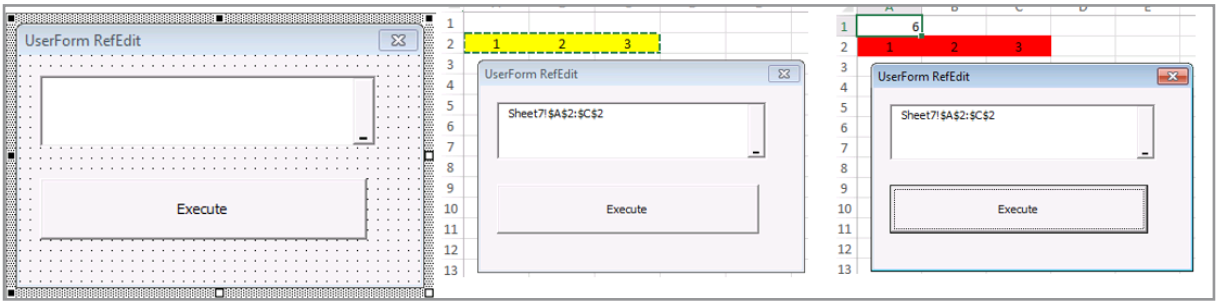


Figure 63: Example for RefEdit

The Listing looks like this:

```
Private Sub CommandButton1_Click()
New_Range = RefEdit1.Value
If New_Range <> "" Then
   With Range(New_Range)
      .Interior.Color = vbRed
   End With
End If
Range("A1").Value = WorksheetFunction.Sum(Range(New_Range))
End Sub
```

1.3.15 TOGGLEBUTTON

A **ToggleButton** can be recognized by the symbol  . The toggle buttons are similar to command buttons. It looks like a push-button light switch. That is, if we press the light switch, then the light is on, and when the light switch is not pressed, the light is off. Therefore, this switch has two states. This state of the toggle button can be queried with the value. If it is pressed then the value is True, otherwise the value is False. So pressed = True and not pressed = False.

Now for the examples: In the first example, pressing the toggle button, i.e. Value = True, displays the button caption in the first line as “Pressed” and in the second line as “Color red” in red text, and the entire worksheet should be filled with red.

If the toggle button is not pressed, i.e. Value = False, the button label in the first line is “Not pressed” and the second line is “Color yellow”. It should be displayed in black and the entire sheet should be filled in yellow.

The whole thing is queried with the if-then decision. If value = true, then top section or otherwise bottom section should be executed.

We make most settings in the properties window. Occasionally, however, they are written directly in the VBA code. Normally, all properties that can be seen in the properties window can be directly included in the listing with a preceding point. In our example we have .Caption or .ForeColor.

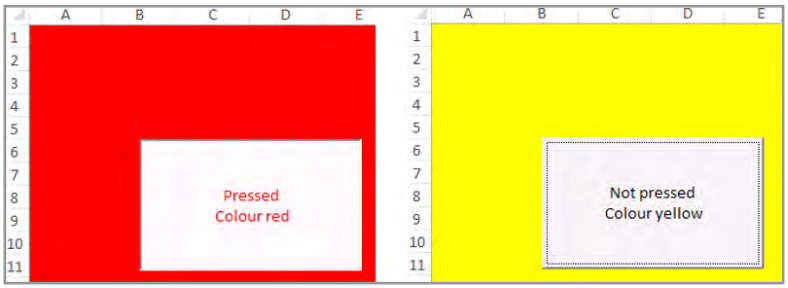


Figure 64 Example ToggleButton_1

The listing looks like this:

```
Private Sub ToggleButton1_Click()
  With ToggleButton1
    If .Value = True Then
      .Caption = "Pressed" & vbLf & " Colour red"
      .ForeColor = vbRed
        With Range("A:XFD").Interior
        .Color = vbRed
        End With
      Else
      .Caption = "Not pressed" & vbLf & " Colour yellow"
      .ForeColor = vbBlack
        With Range("A:XFD").Interior
        .Color = vbYellow
        End With
    End If
  End With
End Sub
```

The next example is intended to colorize all cells that contain a formula or a comment.

```
Private Sub ToggleButton2_Click()
  With ToggleButton2
    If .Value = True Then
      .Caption = "Cells with Formulas"

        With Cells.SpecialCells(xlCellTypeFormulas).Interior
      .Color = vbRed
        End With
      Else
      .Caption = "Cells with comment"
        With Cells.SpecialCells(xlCellTypeComments).Interior
        .Color = vbYellow
        End With
    End If
  End With
End Sub
```

```
        With Cells.SpecialCells(xlCellTypeFormulas).Interior
      .Color = vbRed
        End With
      Else
      .Caption = "Cells with comment"
        With Cells.SpecialCells(xlCellTypeComments).Interior
        .Color = vbYellow
        End With
    End If
  End With
End Sub
```

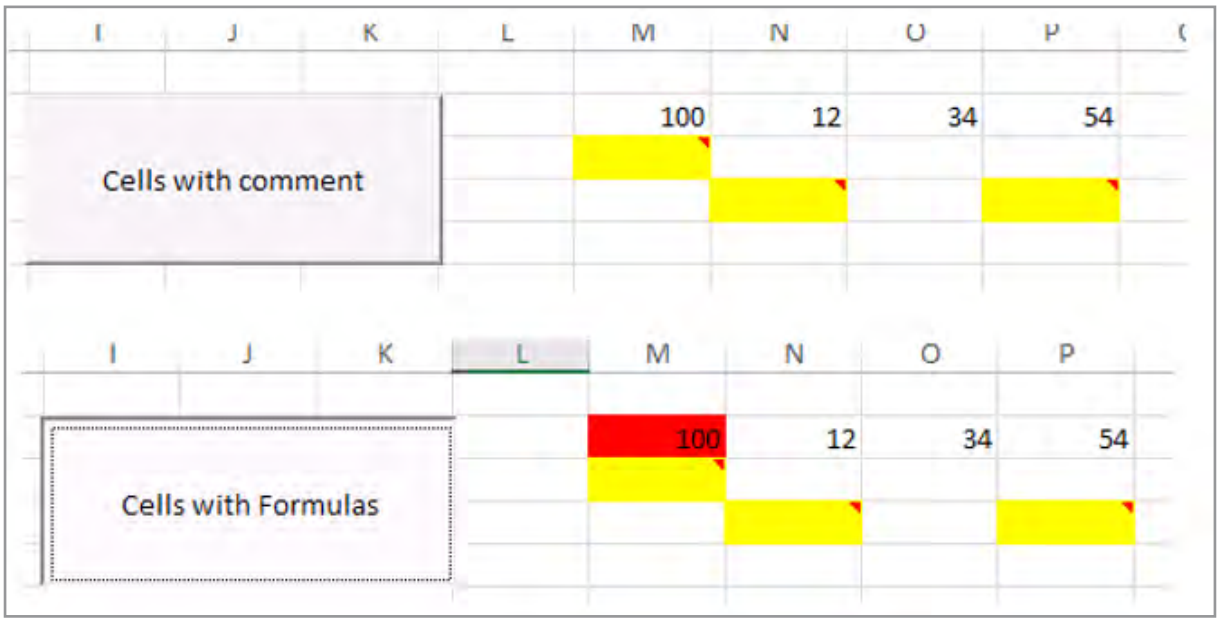


Figure 65: Example ToggleButton_2

1.3.16 IMAGE

An **Image** can be recognized by the symbol  . Using an example, we can better understand how this element works. Here we have pictures in a certain directory. These should be displayed depending on the list box selection.

We have used the following constant here:

- Precise path of the images with LoadPicture:
  - Image1.Picture = LoadPicture (“exact path”)
- Image adjustment on image frame:
  - Image1.PictureSizeMode = fmPictureSizeModeStretch

As usual we first see our UserForm:

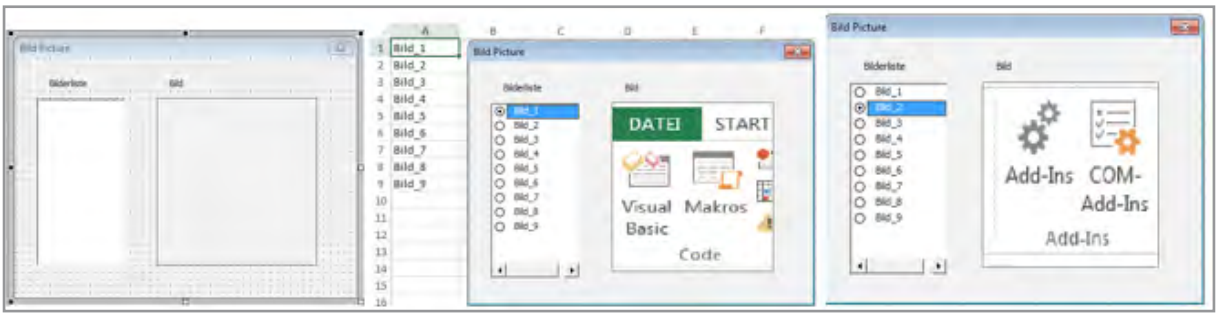


Figure 66: Example for Image

The listing looks like this:

```
Private Sub ListBox1_Click()
Dim Path As String
Path = "C:\Ordner\Unterordner_1\Bilder\"
  If ListBox1.Value <> "" Then
    Image1.Picture = LoadPicture(Pfad & ListBox1.Value & ".gif")
    Image1.PictureSizeMode = fmPictureSizeModeStretch
  End If
End Sub

Private Sub UserForm_Initialize()
  With ListBox1
    .RowSource = "A1:A9"
    .MultiSelect = fmMultiSelectSingle
    .ListStyle = fmListStyleOption
  End With
End Sub
```

BIBLIOGRAPHY

Online Microsoft Developer Network:

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