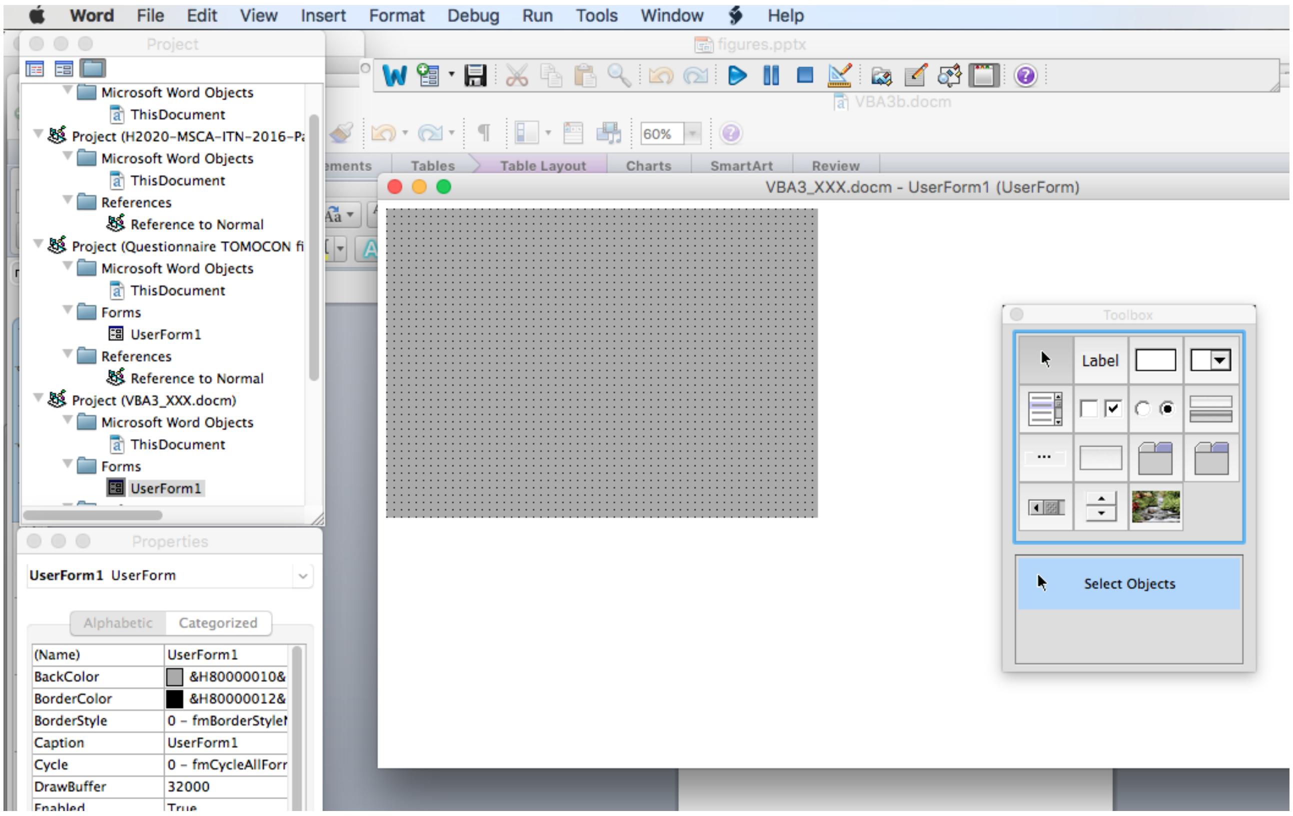
Without GUI, your program is like nothing. Like in other modern languages, VBA allows you to create interface with which you user will interact. So this lab is about introducing popular VBA form objects, how to interact with them, but most importantly, how to manage again Word and Excel actions. The lab introduces new Word objects, but also shows you how it is possible inside one VBA code to create the interaction between different MS applications (e.g. Word and Excel). The MSDN library is good information website to get deeper in VBA language and the (user)forms if you get stuck:
https://msdn.microsoft.com/enus/library/office/ee814735(v=office.14).aspx

INTRODUCTION: VBA userforms

A userform is a synonym for Graphical User Interface (GUI). In Word or Excel, you access to it by opening Visual Basic Editor. If not direct access, create empty macro and edit it to open Visual Basic Editor. Then, on the left panel, you see the current structure of the application. In your project (VBA3_*yourname* .docm) (remember when you start reading or working with Word document to rename it ASAP **VBA3_*yourname*.docm**; again choose the right file extension from drop-down menu, i.e. Macro-enabled extension), click-right on ThisDocument and select **Insert → Userform** . The following template will open (this is called the Object view). Note the Window **Properties** (bottom left) that shows you default properties/attributes of the selected/inserted object (here shows Userform properties) set at **Design time**.

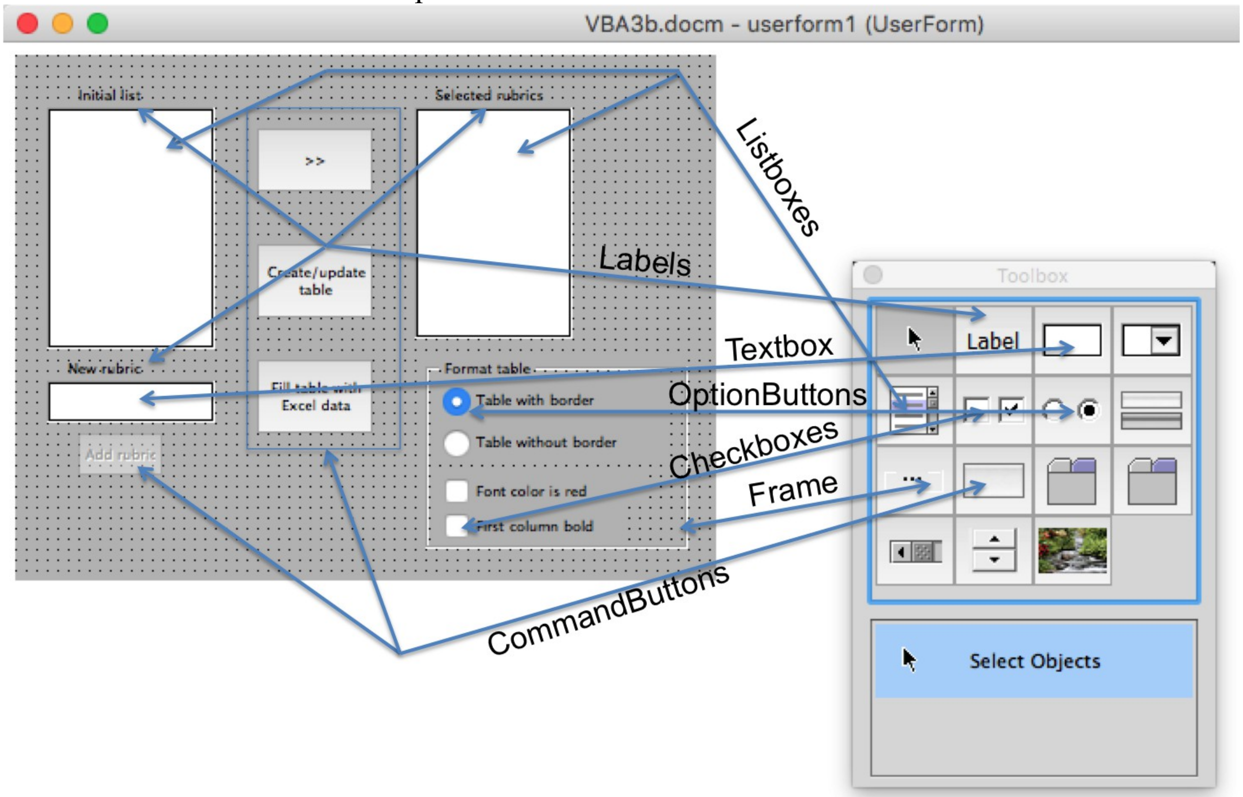


Microsoft Word Objects
ThisDocument
Project (H2020-MSCA-ITN-2016-P
Microsoft Word Objects
ThisDocument
References
Reference to Normal
Project (Questionnaire TOMOCON fi
Microsoft Word Objects
ThisDocument
Forms
UserForm1
References
Reference to Normal
Project (VBA3_XXX.docm)
Microsoft Word Objects
ThisDocument
Forms
UserForm1

UserForm1 UserForm

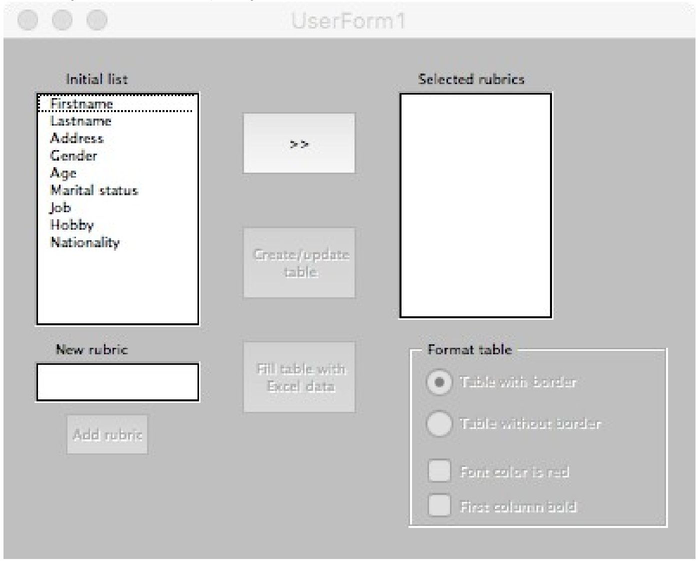
(Name)	UserForm1
BackColor	&H80000010&
BorderColor	&H80000012&
BorderStyle	0 - fmBorderStyleN
Caption	UserForm1
Cycle	0 - fmCycleAllForr
DrawBuffer	32000
Enabled	True

And the one below is the one you have now to design. It contains many **Control** objects: two ListBoxes, 4 CommandButtons (3 enabled, 1 disabled at design time), 1 TextBox, 3 Labels, 1 Frame in which are inserted 2 OptionButtons and 2 CheckBoxes.



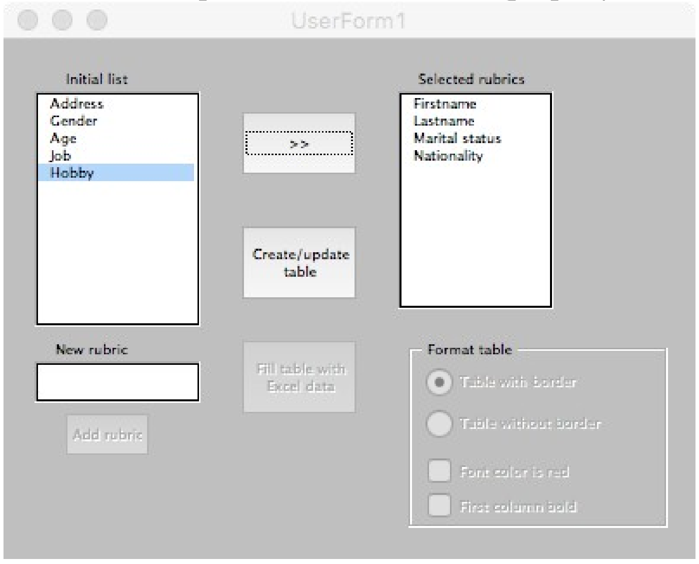
However, to help design the full code, here is a sequence of actions illustrated with screenshots.

<u>**Action 1 (initialization):**</u>At runtime, the **Userform** is **initialized** and it looks like below before any action has been performed by the user (so you see the left ListBox is filled with some data).



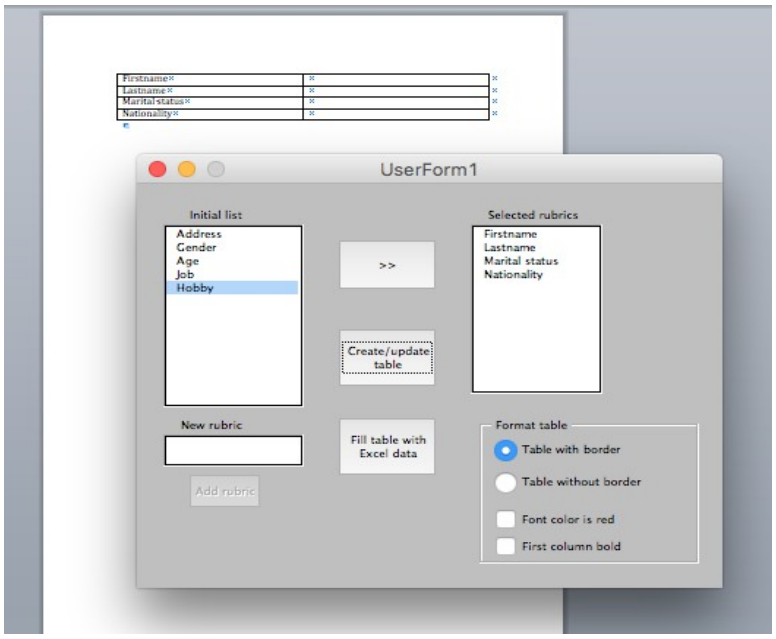
<u>Action 2 (selection):</u> when the user selects an item from the left ListBox and **click** the ">>" CommandButton, two actions occur:

1. the right listbox is populated with the selected items, which are removed from the left listbox
2. The commandbutton "Create/update table" is enabled (property **Enabled** set to **True**)

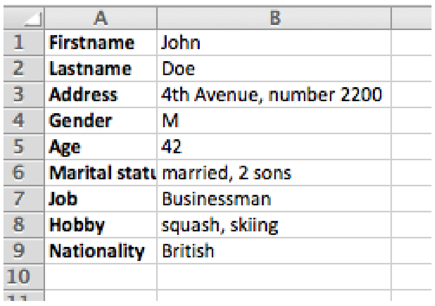


<u>Action 3 (create table):</u> when the user **clicks** the "Create/update table" button, 3 actions occur:

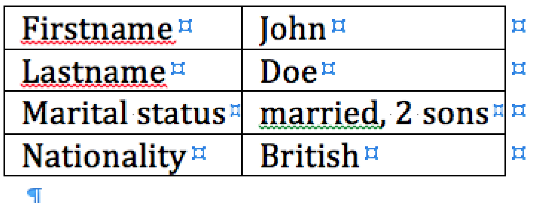
1. a table of 2 columns and a N rows, which corresponds to the number of *selected rubrics* is added to the blank Word document.
2. The objects (OptionButtons and checkboxes) in the frame "Format table" are enabled for interaction.
3. The commandbutton "Fill table with Excel data" is enabled for interaction.



Action 4 (Fill table): when the commandbutton "Fill table..." is **pressed/clicked**, the data located in the column "B" of Excel document are added in the Word table for the corresponding categories, provided it is verified that a data exists for such item.



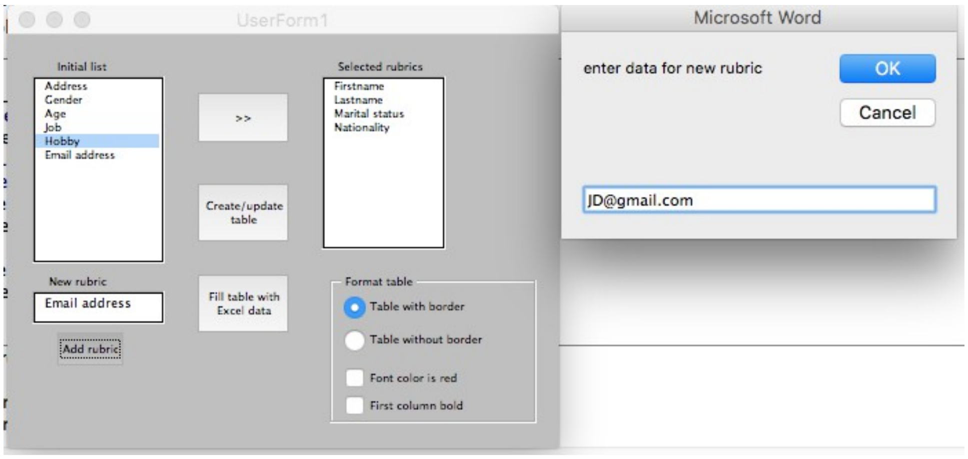
	A	B
1	Firstname	John
2	Lastname	Doe
3	Address	4th Avenue, number 2200
4	Gender	M
5	Age	42
6	Marital statu	married, 2 sons
7	Job	Businessman
8	Hobby	squash, skiing
9	Nationality	British
10		



Firstname	John
Lastname	Doe
Marital status	married, 2 sons
Nationality	British

Action 5 (add new rubric): when a new rubric is entered in the TextBox, the commandbutton “Add rubric” is enabled. When the button is clicked, 2 actions occur:

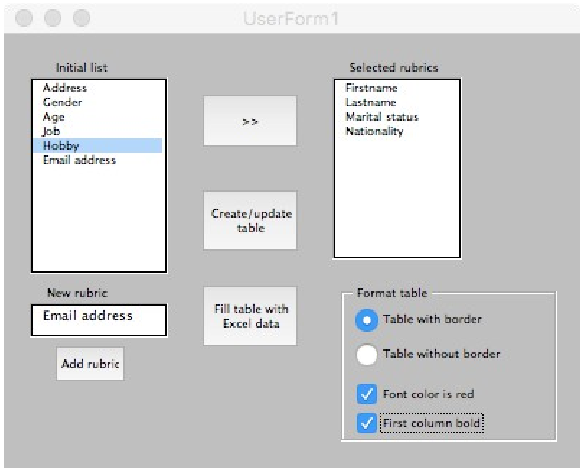
1. the item is added to the left listbox
2. an input box is opened and ask for the corresponding data to be added automatically to Excel database for the new rubric (rubric also added in Excel as last category in column “A”).

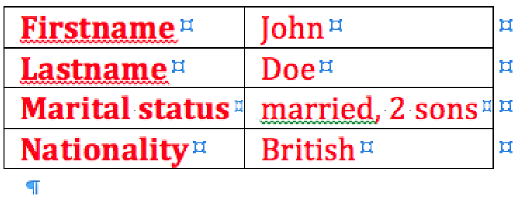


enter data for new rubric

JD@gmail.com

Action 6 (format table): choice of a table with or without borders can be chosen from either selected OptionButtons “Table with border” or “Table without border”. Also, the table can have the first column bold or its text in red font if user chooses to **click** on the corresponding checkboxes.





Firstname	John
Lastname	Doe
Marital status	married, 2 sons
Nationality	British

Exercise 1: prepare Userform

Prepare your project so as it looks like the one above:

- Open a blank word document and save it As :VB3_yourname.docm
- Open Visual Basic Editor and insert empty Userform
- Add the objects mentioned above. Whenever you insert object, change its default name in the **Properties** window to something meaningful, e.g. ***Lbinit*** instead of ***ListBox1*** for the left ListBox in the Userform, or ***CBSel*** instead of ***CommandButton1*** for top CommandButton on Userform. Change also the **Captions** as displayed on the Userform. The table below will help you setting (almost) everything.
- The **CommandButton** with displayed caption "Add rubric" should have property ***Enabled*** set to ***False.***
- Don't forget to often save your file.
- If you want, you can use another color background for the userform or commandbuttons. Also, the position of all objects can be changed. Make your display unique from other students! What is important is that the set of actions is done correctly.
- Create Excel file VB3_yourname.xlsx (save it in same folder as .docm file) and enter the data shown in action 4 above (replace data in column B with the data of your choice if you wish).

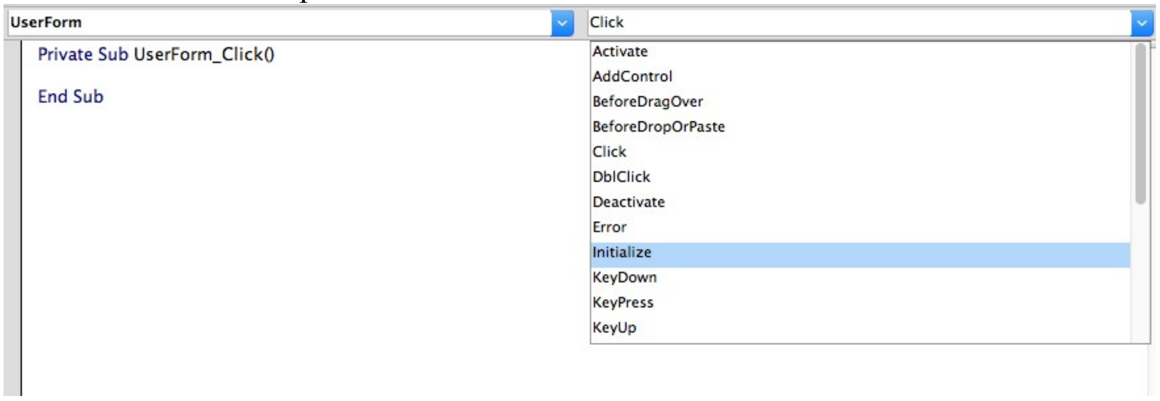
Control	**Name**	**Caption / remark**
ListBox	ListInit	Left listbox in Userform
ListBox	ListChoice	Right listbox in Userform
CommandButton	CBSel	>>
CommandButton	CBCreateTBL	Create/update table
CommandButton	CBFillTBL	Fill table with Excel data
CommandButton	CBaddRub	Add rubric
Label	Label1	Initial List
Label	Label2	Selected rubrics
Label	Label3	New rubric
TextBox	TBadd	
Frame	Frame1	
OptionButton	OBBorderYes	Table with borders
OptionButton	OBBorderNo	Table with no border
Checkbox	CBoxBold	First column is bold
Checkbox	CBoxFTCol	Font color is red

VBA TOOLBOX OBJECTS – PROPERTIES, METHODS AND EVENTS

So far, you could noticed that we can changed some properties of objects at design time(e.g. Caption, Enabled...). Of course, we can changed them also at runtime, that is when the program is executed. For instance, you can easily understand that we cannot add a new rubric to our small Excel database before the name of the rubric is entered in the TextBox. When a name is entered, then the CommandButton is enabled (so its property value set to **True**). This can only be done at runtime, because it depends on the user's action/interaction with the GUI. This actually introduces a new functionality of Control objects: as normal VBA objects, they are represented by attributes and methods. However, another functionality is **Events:** whenever the user will interact with a control (e.g. Click on it (mouse left/right click), drag-drop an object on it, type something in it...), an action of block of statements will be executed. It looks like a procedure structure, except that it is bounded to specific user actions. We will see that all other this lab.

Exercise 2: get ready with action 1

The **<u>action 1</u>** is about Initializing the Userform. To make it appear in your code section, double-click anywhere on the free space in the form. A template **Private Sub UserForm_click()** should be added. This is not the Event we are interested in (we will not click on the form!). However, what is important is to look at the top of the form module.



```
Private Sub UserForm_Click()

End Sub
```

You see that the right combo box shows all possible Events associated with the control shown on the left combo box. Click on Initialize and the template will be added to your code.
More than opening the Userform and make it accessible to the end-user, this event contains a set of important codelines to be declared inside its body part:

- the items in ListInit are added. This is done by calling the method **ListInit.AddItem *string_variable*** (e.g. ListInit.AddItem “Firstname”)
- CommandButtons ***CBCreateTBL***, ***CBFillTBL***, ***CBaddRub*** and OptionButtons/Checkboxes in ***Frame1*** are disabled (their **Enabled** property set to **False**)

Run your project to see if it looks like the screenshot shown under action 1. Don't forget to save your project!

Exercise 3: realize action 2

Action 2 introduces more how to handle Listboxes. When you look at the action description, the click action on the commandbutton ***CBSel*** (**CBSel_click**) gives the feeling that an item from **ListInit** is moved to **ListChoice**. Actually, three main actions occur:

1. An item in ***ListInit*** is selected. We can track the index of the item by calling the property **ListIndex.** If an item has been selected, its index value is automatically assigned to **ListIndex**, otherwise **ListIndex** is set to -1.
2. We copy the item value to the right ListBox ***ListChoice***. This is done by again calling the AddItem method, but *string_variable* is replaced here by **ListInit.List(ListInit.ListIndex).**
3. The selected item is removed from ListInit by calling the **RemoveItem(*Index*)** method, where ***Index*** is replaced again by **ListInit.ListIndex.**

Another action is executed when calling **CBSel_click**, that is the commandbutton ***CBCreateTBL*** is enabled.

Apply the instruction above (first double-click on the ***CBSel*** button to automatically include the template of **CBSel_click** to the Code window). Check the value of **ListIndex** and take decision: if -1, set the index value to 0 so as always the first element in the item list is selected (i.e. item "Firstname"). It is highly advised to declare a variable to store the value of the item index.
Run the project, click on few ***ListInit*** items and the ***CBSel*** button sequentially and see if the item movement is simulated correctly.

Note: A ListBox in its standard form looks like a 1D array, where the first index value is 0. We can easily know how many items are in the list by calling the **ListCount** property. The value is returned by the property **List(*index*)** for the given *index* value: 0- first item; (**ListCount**-1)- last item in list.

Exercise 4: realize action 3

Action 3 corresponds to the **CBCreateTBL_click** event. When such event occurs, following actions occur:

- The content in the Word document is deleted, using the statement
  ActiveDocument.Range.Delete
- Add table object to document using following method
  `Set TB=ActiveDocument.Tables.Add(Range,NumRows,NumColumns)`
  Declare a Table object as a module-level object variable (below we call it TB). Set the *Range* variable empty by using **ActiveDocument.Range(0,0)**. The number of rows corresponds to the number of selected items in ***ListChoice*** and the number of columns equal 2.
- Insert in the first column the selected rubrics of ListChoice using Table method
  TB.Cell(*Rowindex*,*Columnindex*).Range.Insertafter *string_variable*
  Use loop to insert the rubric names, *Rowindex* being the variable to be iterated. Optional: you can format the table so as the column widths fit the cell content. For that, use the following method
  TB.AutoFitBehavior (*wdAutoFitContent*)
- CommandButton ***CBAddRub*** and control objects in ***Frame1*** are enabled for interaction

Create the VBA code corresponding to the above actions and see the result. If you have the Word document in the background, you can see if the first column of the table is filled with rubric items. Of course, save your project before execution. We never know what can happen!

Exercise 5: action 4 or communication between Word and Excel applications

The second step of the table filling (2nd column) consists in accessing Excel data and for the selected items, get the corresponding elements from Column B. This involves that Excel objects are declared in the VBA project. Because the current environment is driven by the Word application, some adjustment needs to be performed. First, you need to allow your project to access Excel objects. For that, click on the **Tools** tab in the VBA editor and select **References...**Then click on the checkbox "Microsoft Excel Object Libraries".
Having this setting done, your are ready to invoke Excel objects in your code. Since, you will have to access data from Excel spreadsheet, you will need to open your Excel document created in Exercise 1. To do so, invoke the following code. Preferably, write it in the **UserForm_Initialize** event (then your Excel document will be opened when Userform is loaded/created).

```
Dim path As String
' Open Excel and workbook where data are
path = CurDir 'always good practice to have files in same folder
' But in any case, retrieve it.
path = path & "\VBA3_yourname.xlsx" 'replace yourname accordingly
Set xlapp = CreateObject("Excel.Application") 'Launch Excel
xlapp.Visible = True 'make it visible on screen
Set wrkbk = xlapp.Workbooks.Open(path) 'Open Excel file defined by variable path
```

You will need also to create 2 Excel objects: **xlapp** and **wrkbk**, preferably as module-level variables. However, to not confuse the VBA interpreter, declare them with data type starting with Excel:

```
Dim xlapp As Excel.Application
Dim  wrkbk As Excel.Workbook
```

The Excel document being opened, most of the current **action 4** is linked with clicking on the commandbutton ***CBFillTBL***. By doing so, the following code will be executed:

- for each item of the first column of the table, you need to check if it is present in the Excel spreadsheet (column "A"). Because the item can be composed of more than 1 word (e.g. "Marital status"), you need to concatenate these words into a string variable to make the string comparison possible with Excel data. This concatenate can look as follows

```
TB.Cell(i, 1).Select 'select a cell of the first column of the table
tpnw = Selection.Range.Words.Count - 1 'calculate number of words, always 1 more
'for some reason
str1 = "" 'initialize str1 variable with empty string
For k = 1 To tpnw
       str1 = str1 & Selection.Range.Words(k) 'do concatenation
Next k
```

- Compare each string with the ones in column "A" of the Excel document. If you use a For loop to check all strings, you will need to know what is the index of the last row of the data range. For that the following code does the job

```
Set sh = wrkbk.Worksheets(1) 'assume worksheet object sh is declared
r = sh.Range("A1", sh.Range("A1").End(xlDown)).Rows.Count
```

- Using **strcomp(*string1*,*string2*))**, one can check if the selected rubric is in the database to retrieve the corresponding data. Remember that cell data can be easily retrieved in Excel using the **Cells** object, e.g. s**h.Cells(i,2).Value**

Include and modify if necessary the codes above and execute the project action by action to see if the (first) final result of filling the table in Word works!

Exercise 6: allowing adding new rubric and data (action 5)

The Textbox ***TBadd*** is central to this action. When a text of a new rubric (e.g. "Email address") is entered in the textbox (so its content is **Changed** and is stored in the property **Tbadd.Text**), the commandbutton ***CbaddRub*** is **Enabled**. Then, when **clicked,** the new item is both added to **ListInit** and in the first available cell in Column "A" (so below the last element of the column. Its index value should be calculated. However you know the index of the last element, since calculated for <u>action 4.</u> So peace of cake!). Furthermore, as described earlier, you should use an **Inputbox** call to ask the user about the corresponding data (e.g. a valid email address) that is directly assigned to the Excel document. The corresponding code of the latter action is:

```
sh.Cells(index, 2) = InputBox("enter data for new rubric " & sh.Cells(index, 1))
```

where index stands here for the row index where the data is entered.

The hardest stuffs are behind you. That shouldn't take you too long to do this task!

Exercise 7: aesthetic Table formatting to understand how OptionButtons and CheckBoxes work (action 6)

In the Frame ***Frame1*** are 2 new types of Control objects: OptionButton and Checkbox. While any of the Checkboxes can be clicked, only one OptionButton can be clicked at design/runtime. You will notice this when you will run the program that you cannot have both OptionButtons selected.

We use in this project the option of turning On/Off the borders of the Table in the Word document. For that, we can simply set the ***ColorIndex*** of the ***Borders()*** property set to wdBlack (On) or wdWhite (Off). ***Borders*** take input arguments corresponding to the type of line (left/right/top/bottom border of edges, horizontal/vertical for inner borders), which are popping out when the left bracket "(" is typed.

In the case of the checkboxes when they are selected, the text will be red (**TB.Range.Font.ColorIndex = wdRed**) and the font of the 1st column will be bold

```
Private Sub CboxBold_Click()
TB.Range.Columns(1).Select
Selection.Font.Bold = True
End Sub
```

A bit is given. Do the rest! And Test!

Exercise 7: shut down!!!!!!

As you have initialized some variables when the Userform was executed, you can also perform some actions when the application is terminated. For instance, it is a good practice to remove created objects from memories using the `Set ObjectName=Nothing` statement and also to close Excel, since only the result in Word interests us (but we can also check if new data has been added to Excel as well, so we can decide to keep it opened). An example of code could look like this:

```
Private Sub UserForm_Terminate()
xlapp.Quit
Set xlapp = Nothing
Set TB = Nothing
End Sub


'----------------------------------------THE END-------------------------------
```

REMARK: IF YOU MANAGED TO COMPLETE THIS LABORATORY AND UNDERSTOOD ALMOST EVERYTHING THEN I AM CONFIDENT THAT YOU WILL SUCCEED WITH YOUR PAIR-PROJECT. IN ANYCASE, YOU WILL NEED TO LEARN BY YOURSELVES ANY TOPICS NOT COVERED BY THE THREE VBA LABS.