TechMentors

Helping you get the most out of technology

Excel VBA UserForms

Student Manual

www.theTechMentors.com
info@theTechMentors.com
Revised May 2025

1. Introduction to Excel VBA UserForms 3
   A. What is a UserForm? 4
   B. What are the benefits of a UserForm? 6

2. Creating and Programming UserForms 7
   A. How do I add a UserForm to my workbook 8
   B. Add controls to the UserForm 9
   C. Five-step UserForm Creation Process 11
   D. Set Properties 12
   E. Exercise: Decipher the macro's code 14
   F. Writing code and Event Driven Programming 15
   G. Test and debug code 17
   H. Exercise: Begin creating your UserForm 18

3. Text Boxes, Labels, Lines, Boxes and Tab Order 20
   A. What is a Text Box? 21
   B. What is a Label? 23
   C. How can I align objects horizontally and vertically? 25
   D. Exercise: Add labels and text boxes 27
   E. Tab Order 29

4. List boxes and Combo boxes 30
   A. What is a List box? 31
   B. What is a Combo box 33
   C. How can I populate a list box or combo box via code? 35
   D. How can I populate a list box or combo box from a spreadsheet? 36
   E. Exercise: Add a combo box for Departments 37

5. Spin buttons and Scroll bars 40
   A. What is a Spin button? 41
   B. What is a scroll bar? 42
   C. Exercise: Add a spin button to your UserForm 43

6. Working with multiple UserForms 45
   A. How do I duplicate a UserForm? 46
   B. How can I export a UserForm to a file? 47
   C. How can I import a UserForm from a file? 48
   D. How can I remove a UserForm from an Excel workbook? 49
   E. Exercise: Import a UserForm Calendar 50
   F. How can I add code to work with multiple forms 52

G. Exercise: Add controls and code to work with the calendar form 55

7. **Check boxes and option buttons 57**

A. What is a check box?.. 58
B. What is an option button? 59
C. What is a frame control? 60
D. Exercise: Add a check box to your UserForm. 61
E. Exercise: Add a group of option buttons to your UserForm. 62

8. **Loading and Saving UserForm Data in Excel.................... 63**

A. How can I populate my UserForm from my Excel worksheet?.................... 64
B. Exercise: Populate the Employees form with Excel data 67
C. How can I write code to save UserForm data to Excel?............................. 72
D. Exercise: Add and save data from the Employees form 74

9. **Pictures and Images 76**

A. What is an image control?.. 77
B. Do other controls have a picture property?... 79
C. Exercise: Enhance your UserForm with images and pictures...................... 80

10. **The RefEdit controls... 88**

A. What is a RefEdit control?... 89
B. Exercise: Add a RefEdit control to your UserForm 90

11. **Open Lab 94**

1. Introduction to Excel VBA UserForms

Complete this unit and you'll be able to:

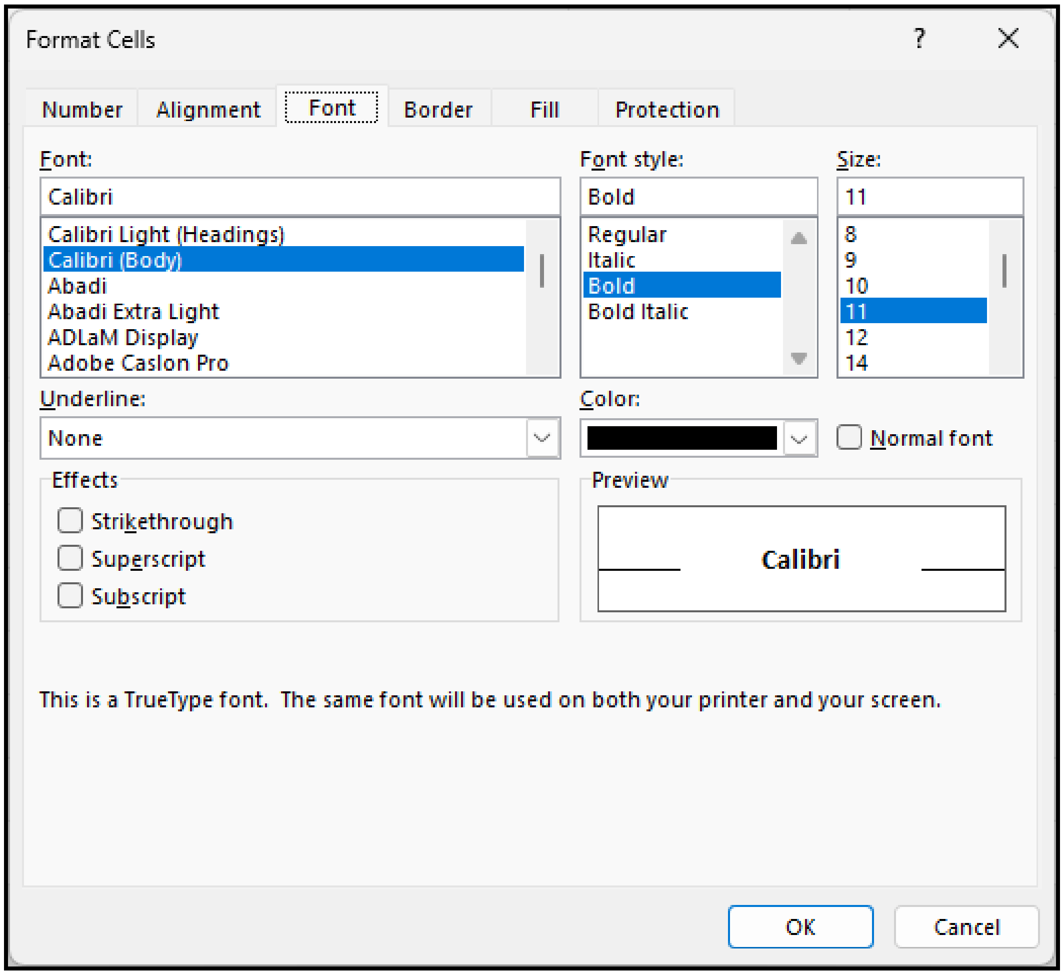
A. Describe the features and benefits of UserForm

B. Create a UserForm

C. Add controls to a UserForm

A. What is a UserForm?

Figure 1 below is a screen shot of Excel's Format Cells dialog box, which, as you know, has multiple tabs that let users easily pick and choose from various number formats, alignment settings, fonts, borders, fills, and protection options. This dialog box is a form that was created by Microsoft.

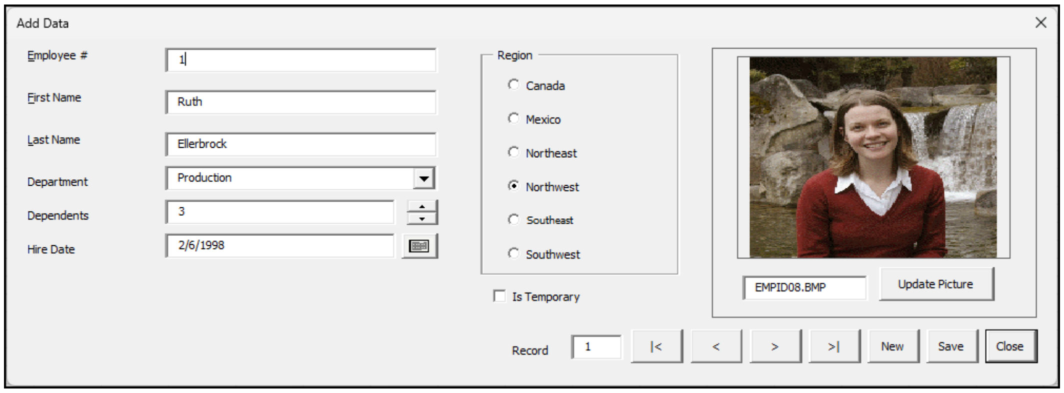


This is a TrueType font. The same font will be used on both your printer and your screen.

Figure 1

As a VBA Programmer, you can create similar dialog boxes which are called UserForms. Like the ones created by Microsoft, your UserForms can have multiple tabs, text boxes, drop-down menus, check boxes, labels, and command buttons. When combined with some VBA code your UserForm will let your users easily input, modify, print, save, and manage the data stored in your spreadsheets.

In this course you will learn how to create UserForm like the one shown in Figure 2.



Add Data

Employee #	1
First Name	Ruth
Last Name	Ellerbrock
Department	Production
Dependents	3
Hire Date	2/6/1998

Region

- Canada
- Mexico
- Northeast
- [✓ Northwest]
- Southeast
- Southwest

Is Temporary

EMPID08.BMP

Figure 2

B. What are the benefits of a UserForm?

UserForms can save you and your organization a lot of time and trouble. Instead of dealing with the chaos that comes from multiple users each creating and editing a spreadsheet in their own way, you can instead provide a controlled environment that automates and ensures consistency among your various users.

You can create UserForms for a variety of solutions including:

- Data entry forms that organize input and reduce errors.
- Dashboards that help users manage and visualize key metrics.
- Interactive tools that provide the right calculations and facilitate decision-making, and/or data analysis.
- Custom wizards to that use VBA to guide users through a complex multiple step process.

2. Creating and Programming UserForms

Complete this unit and you'll be able to:

A. Add a UserForm to your workbook

B. Add controls to the UserForm

C. List the five steps of the UserForm design process

D. Set properties

E. Exercise: Decipher the macro's code

F. Write VBA code

G. Test, debug and fine-tune your form and its code

H. Exercise: Begin creating your UserForm

A. How do I add a UserForm to my workbook

To add a UserForm to your form, do the following:

1. Start Excel and open the desired workbook.
2. Open the VBA Editor by pressing **Alt** + **F11**.
3. Insert a new UserForm by selecting **Insert** > **UserForm** from the menu.

The next step is to add controls to the form, which is covered in the next topic.

B. Add controls to the UserForm

Controls, as shown below, are objects, such as labels, text boxes, and command buttons that you can add to your UserForm.

Name	Icon	Prefix	Description
Label	A	lbl	Creates a text label that the user cannot change.
TextBox	abl	txt	Creates a box that holds text that the user can modify.
ComboBox	▤	cbo	Creates a drop-down menu. It's called a combo box because it combines features of a text box and ListBox.
ListBox	▤	lst	Creates a menu of text items from which the user can choose.
CheckBox	☑	chk	Creates a box that can be checked or unchecked by the user.
OptionButton	◉	opt	Used to create a series of choices (radio buttons) from which the user can only select one option. Should be used inside a frame.
ToggleButton	◧	tog	Creates a button that can be togged on or off.
Frame	⬚	grp	Creates a graphical / functional grouping for other controls.
CommandButton	ab	cmd	Creates a button that the user can click to execute a command.
TabStrip	▭	tab	Used to define multiple pages for the same set of controls.
MultiPage	🗀	mpg	Presents multiple pages of various controls.
ScrollBar	⬍	scr	Creates a horizontal or vertical bar that the users can scroll up/down or left/right.
SpinButton	⬍	spn	Creates a spinner control that users can click to increase or decrease a value.
Image	🖼	img	Displays an bitmap, icon, or metafile image.
RefEdit	▣	ref	Displays a control that lets users select a range of cells.

Table 1: Controls

To add a control to your form, do the following:

1. Click on the form.

2. If the Toolbox, shown in Figure 3 is not displayed, choose **View** > **Toolbox** from the menu.

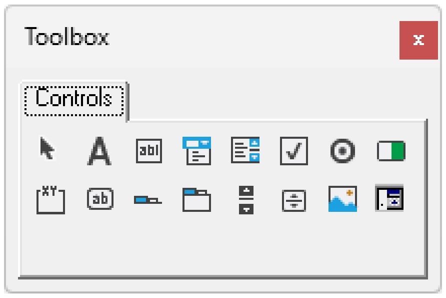


Figure 3

3. Click on the control you want to add.

4. Click in the desired location on the form and then drag the mouse to size it as you like.

Or you can simply click in the desired location and the desired control, with a predetermined size, will be added to the form.

You can **resize** the control after it is drawn, by clicking on the white square and then dragging with the double headed black arrow.

You can **move** the control by clicking between the white squares and then dragging the control to the desired location with the four-headed black arrow.

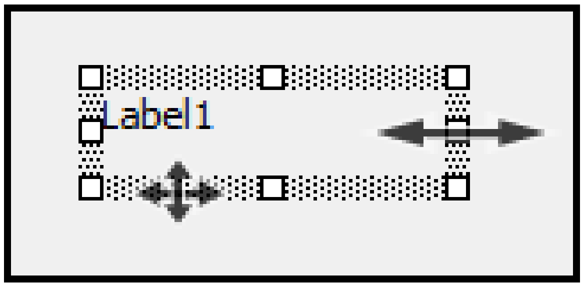


Figure 4: The double-headed black arrow will resize the control. The four-headed black arrow is used to move the control.

C. Five-step UserForm Creation Process

When designing a UserForm you should execute this five-step UserForm design process to create a user-friendly form that works and is easy to maintain.

1. **Plan your form**'s look, feel and functionality.

   Begin this step by documenting the purpose of your form.

   Determine which labels, text boxes, drop-downs, command buttons, etc. will be used to gather and share data.

   Envision on paper where each control will be located. This is especially important for complex forms. For simple forms, you may be able to do this in your head.

   Document the actions you want the user to take. Then determine the reactions the form's code will need to execute.

2. **Draw** the form and its controls.

   This is the step where you create the form (**Insert** > **User Form**) and add the controls based on your plan.

3. **Set properties** for each control.

   The form is a VBA object, and each control is an object. Like all other VBA objects, forms and controls each have properties. Some properties like (Name) need to be set for programming purposes. Other properties, like a label's **`Caption`** and font need to be set for appearance purposes. We will explain how to set properties in topic D. Set Properties.

4. **Write VBA code**.

   Based on the actions the users take, and the reactions you want, you will need to write the VBA code that executes those actions. This will be explained in topic E. Write code.

5. **Test, debug and fine-tune** as needed.

   After writing the code, you will need to test and debug your code. This will be explained in detail in topic F. Test and debug code.

D. Set Properties

Before you can set an object's properties, you must first select the object and then display the Properties Window by selecting **View** > **Properties** from the menu or by pressing **F4**.

Please note the following as shown in Figure 5 below:

A. The top of the Properties Window displays the name of the selected object.

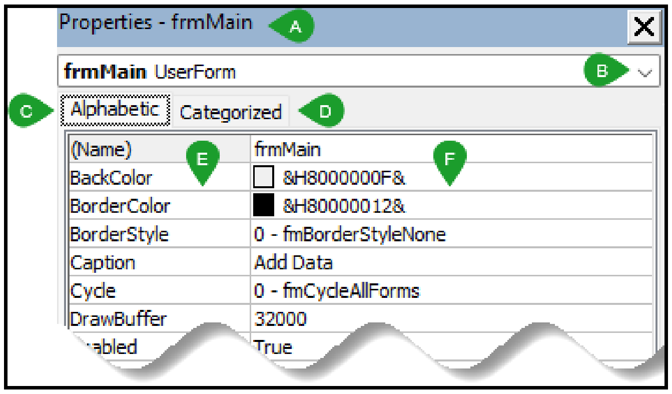
B. The Properties Window has a drop-down menu that when clicked lists the form and its controls. You can use this drop-down menu to pick the object you want to manipulate.

C. The properties window has two tabs: Alphabetic and Categorized. The **Alphabetic tab** lists the properties in alphabetical order with the name property in parentheses, thus forcing it to the top.

D. The **Categorized** tab groups the properties into various categories and then lists them in alphabetical order.

E. The first column in the Properties Window is the name of the property.

F. The second column is the value of the property.
You can use the mouse and/or keyboard to change the value as needed. This will be demonstrated in upcoming exercises.



(Name)	frmMain
BackColor	☐ &H8000000F&
BorderColor	■ &H80000012&
BorderStyle	0 - fmBorderStyleNone
Caption	Add Data
Cycle	0 - fmCycleAllForms
DrawBuffer	32000
abled	True

Figure 5

VBA Object Naming Conventions

One of the most important properties you must set is the **`(name)`** property. Misnaming an object can cause nightmares when it is time to code the form. Therefore, you should follow these naming conventions:

- Start with a **lower-case prefix** that identifies the control's type:
  For example you can use *txt* for text boxes, and *lbl* for label.
  You can find a list of suggested prefixes in Table 1: Controls above.

- **Be descriptive**: The name should include a description of the control's purpose or function. This will make it memorable. Having a bunch of objects named *CommandButton1*, *CommandButton2*, and *CommandButton3* will cause you, the programmer to waste time trying to remember which object is which. In contrast, names like cmdNew, cmdSave, and cmdClose will make it very easy to remember the names of the New, Save and Close command buttons.

- **No spaces allowed**: The name must not have any spaces because names with spaces can cause problems when you write code for the form and its controls.

- **Be case Sensitive:** Because the control's descriptive part may need to include multiple words (example First Name) but should not include spaces, most programmers will make all letters lowercase except the first letter of each word (FirstName). This is called Camel Case because when combined with the prefix the upper case letters will be like the hump on a camel's back (txtFirstName).

Writing code to refer to a Control by its name.

Because each control on a form is considered a child of the form, you can refer to the control in the code window by entering the keyword **Me** then a period, and then the name of the control. Example: `Me.txtLastName.`

However, when referring to a control from a macro or from another form, you will need to refer to the name of the form, a period, and then the name of the control. Example: `frmEmployee.txtLastName`.

E. Exercise: Decipher the macro's code

Examine the two macros below. Which one is easier to understand? Which one would you rather debug? Why?

```
Private Sub GetDataFromForms
    Range("A2") = "Sales"
    Range("B2") = Form1.ComboBox1.value
    Range("C2") = Form2.TextBox2.vaue

    Range("A3") = "Operations"
    Range("B3") = Form1.ComboBox3.value
    Range("C3") = Form2.TextBox4.value

    Range("A4") = "Accounting"
    Range("B4") = Form1.ComboBox6.value
    Range("C4") = Form2.TextBox5.value
End Sub
```

Figure 6: Macro 1

```
Private Sub GetDataFromForms
    Range("A2") = "Sales"
    Range("B2") = frmDepartment.cboSalesDepartmentManager.value
    Range("C2") = frmBudget.txtSalesBudgetFY2025

    Range("A3") = "Operations"
    Range("B3") = frmDepartment.cboOperationsDepartmentManager.value
    Range("C3") = frmBudget.txtOperationsBudgetFY2025

    Range("A4") = "Accounting"
    Range("B4") = frmDepartment.cboAccountingDepartmentManager.value
    Range("C4") = frmBudget.txtAccountingBudgetFY2025
End Sub
```

Figure 7: Macro 2

F. Write VBA code

Before you can view and write the code for your form, you will want to display the Project Explorer by selecting **View** > **Project Explorer** from the menu or by pressing **Ctrl R**.

As shown in Figure 8 the Project Explorer displays the name of the workbook project (with its name in parenthesis) and its worksheets (Sheet1, Sheet2, etc.) with the names of the worksheets in parenthesis (Employees, Depts, Sheet1). It also displays the names of the forms and the modules for the selected project.

The project explorer also displays these three buttons which are described in reverse order below:

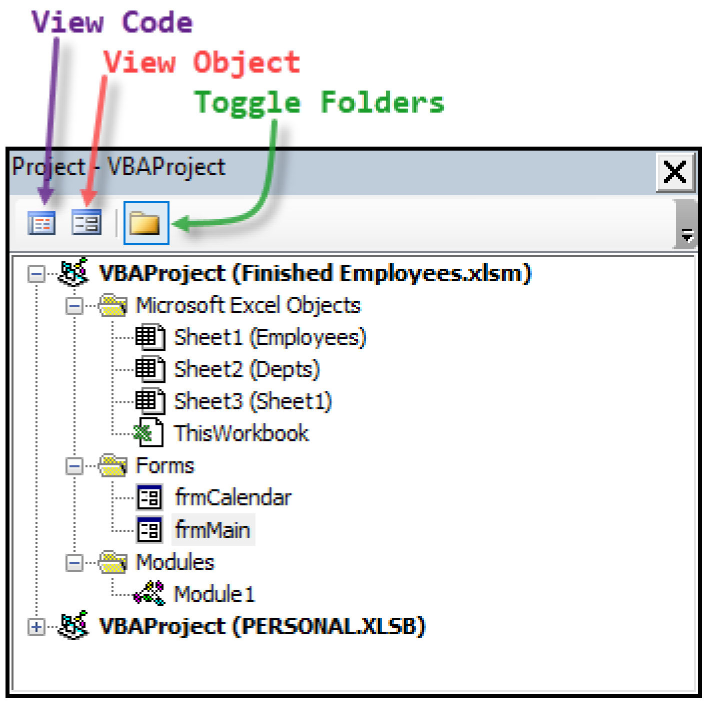


Figure 8

Toggle Folders

When this toggle button is clicked on, all the worksheets, forms, and modules will be displayed in their respective folders. When off, no folders are displayed and these objects are listed in alphabetical order.

View Object

The View Object button will switch back to the form design window which displays the form object and its controls.

View Code

When you click the View Code button, the display will switch to the code window for whichever form or control you have selected. This code window has the same features of the code window used to create and edit VBA macros.

You can also view the code window by simply double clicking on the desired form or control.

Event Driven Programming

Event Driven Programming is a paradigm where a program's execution is determined by events that are usually caused by a user or some other external occurrence. VBA is an Event Driven Programming language. The events are the actions the occur such as when a user clicks a button or changes a value.

For example, a command button has a click event. The statements you write for the click event are then executed when the user clicks that button.

Writing code for an event is similar to writing VBA code for an Excel macro. As shown below, both may begin with the word **Public** or **Private**. They also include the word **Sub**, which is followed by the name of the macro or event, which is then followed by parenthesis. Next comes the statements that get executed, and finally the code finishes with the words End Sub.

Macro	Event

`Public Sub GoToA1()`
`    Range("A1").Select`
`End Sub`
	`Private Sub cmdClose_Click()`
	`    Unload Me`
	`End Sub`

Remember in a macro the programmer decides the name of the macro. But an event sub-procedure's name, as shown above, is a combination of the object's name `cmdClose`, an underscore `_`, and the name of the event `Click()`.

G. Test, debug and fine-tune as needed

After you have planned and created your form and its controls, after you have set their properties, and written code for their events, you are ready to test the form. Simply select the form and then choose **Run** > **Run Sub / User Form** from the menu. You can also run the form by clicking the run button ▶ on the toolbar or by pressing **F5** on the keyboard.

Unfortunately, bugs will often occur. Your code will therefore need some fine tuning. This fine tuning may need to be done multiple times until you are satisfied that it works the way you expected.

Break Points

To aid in the debugging process you may want to add a break point to your code. A break point is like a red stop light on the VBA highway. It forces the code to stop and allows you to walk through the code step-by-step.

To add a break point position your cursor on the statement where the code should stop, and then press **F9**.

Next run your code. When it gets to the break point it will stop. You can then press **F8** to have Excel execute the next statement. You should then continue to press **F8** to watch Excel perform its actions. When you want to run the code again at normal speed, just press **F5**. Finally, when you are done debugging, you can turn off the break point by returning to the statement that has your break point and press **F9**.

H. Exercise: Begin creating your UserForm

In this exercise you will open an Excel file called Employees.xlsx and then immediately save it as an Excel Macro-Enabled Workbook, add a simple user form with a command button, set some properties, add some code, and finally test the form.

1. From the Exercise files folder, open the **Employees.xlsx** file.

2. Press **F12** to save the file, change the file type to an **Excel Macro-Enabled Workbook (*.xlsm)** file, and then click Save.

   Why do you have to save the file as a Macro Enabled Workbook? Because regular Excel .xlsx files are incompatible with UserForms and VBA code.

3. Press **Alt F11** to go to the Visual Basic Editor.

4. From the menu choose **Insert** > **UserForm**.

5. If needed display the Project Explorer by selecting **View** > **Project Explorer** from the menu, or by pressing **Ctrl R** on the keyboard.

6. If needed display the Properties Window by selecting **View** > **Properties Window** or by pressing **F4** on keyboard.

7. In the Properties Window verify that UserForm1 is the selected object (if not click once on the form).

8. Change the **`(Name)`** property to ***`frmEmployees`***.

9. Change the **`Caption`** property to ***`Edit Employees`***.

10. Click on the User Form again.

11. If needed display the Toolbox by selecting **View** > **Toolbox** from the menu.

12. In the Toolbox click on the command button, and then click on the form.

13. In the Properties Window verify that CommandButton1 is the selected object (if not click once on the button that you just added to the form).

14. Change the **`(Name)`** property to ***`cmdClose`***.

15. Change the **`Caption`** property to ***`Close`***.

16. Double click on the Close button. This should bring up the code window.

17. Edit the code to match the following:

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

18. Press **F5** to run the form.

19. Click the **Close** button.

20. Save the file.

3. Text Boxes, Labels, Lines, Boxes and Tab Order

Complete this unit and you'll be able to:

A. Describe the features and uses of a text box

B. Describe the features and uses of a label, and how to use a label as a decorative line or box

C. How can I align objects vertically and horizontally?

D. Exercise: Add text boxes and labels to your UserForm

E. Change tab order

A. What is a Text Box?

A text box, as its name implies, is a box that allows the user to view and edit text.

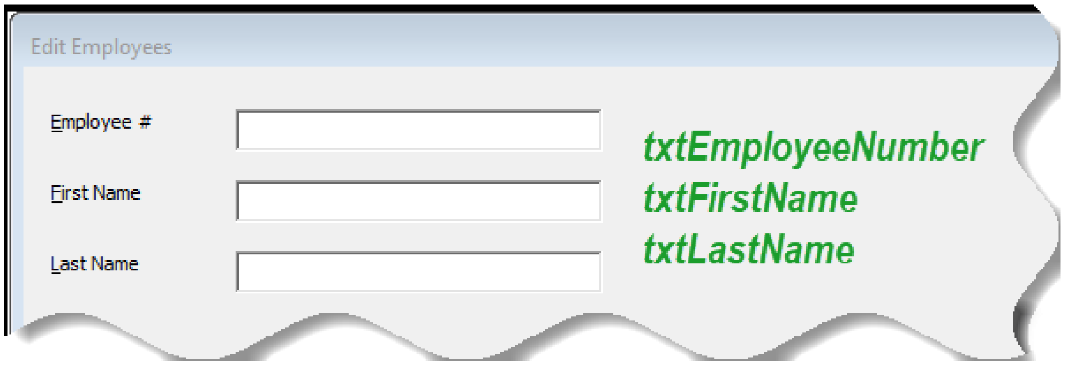
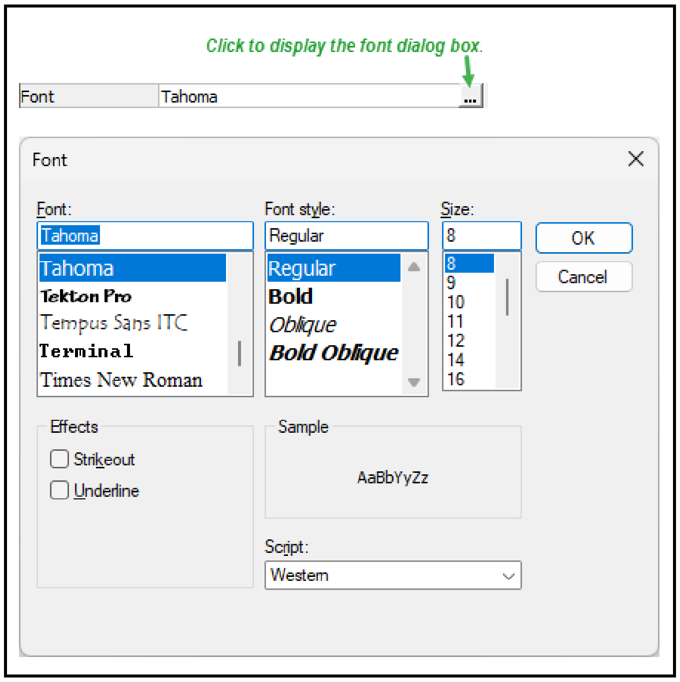


Figure 9: Screenshot showing three text boxes and their names in green.

You can set a default value in the text by entering text in the **`Value`** property.

You, the designer, can change the width of the text box with the mouse or by changing its **`Width`** property. You can also change its height with the mouse or the **`Height`** property.

You can change the font of the box's text by clicking in the **`Font`** property's value box and then clicking the builder button on the right side. As shown below the font property dialog box lets you change the font, font style, size, strikeout, and underline and script settings.



Font: Tahoma

Font: Tahoma
Font style: Regular
Size: 8
Strikeout: unchecked
Underline: unchecked
Script: Western

You can use the **`MaxLength`** property to set the maximum number of characters the user can enter in the text box. This would be useful, for example, when you want to limit a State text box to 2 characters.

By default, the **`MultiLine`** property is set to False meaning the box can only display one line of text no matter how big the box is. Set it to True if you need multiple lines of text. You may also want to verify that the **`WordWrap`** property is set to ***`True`***, otherwise the user will not see the text wrap around until they force a new line by typing Ctrl Enter.

B. What is a Label?

The label control is primarily used to identify objects on the screen. Unlike the text box control, the user will not be able to edit the text of a label object.

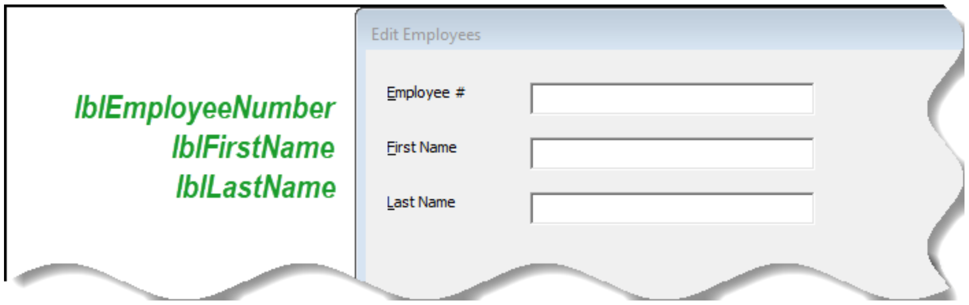


Figure 10: Screenshot showing three labels and their names in green.

The label control can also be used to draw a decorative horizontal or vertical line. To do so, set the **`Caption`** property to blank, and then set the **`BackColor`** property to your desired color. Finally, either use the mouse or set the width and height properties to achieve the desired look.

You can also use the label to create a decorative box around other objects. Begin by setting the **`Caption`** property to blank, set the **`BorderColor`** to your desired color, set the **`BackStyle`** to ***`1 – frmBackStyleTransparent`*** and then size the box with either the width and height properties or by dragging the label's edges with the mouse.

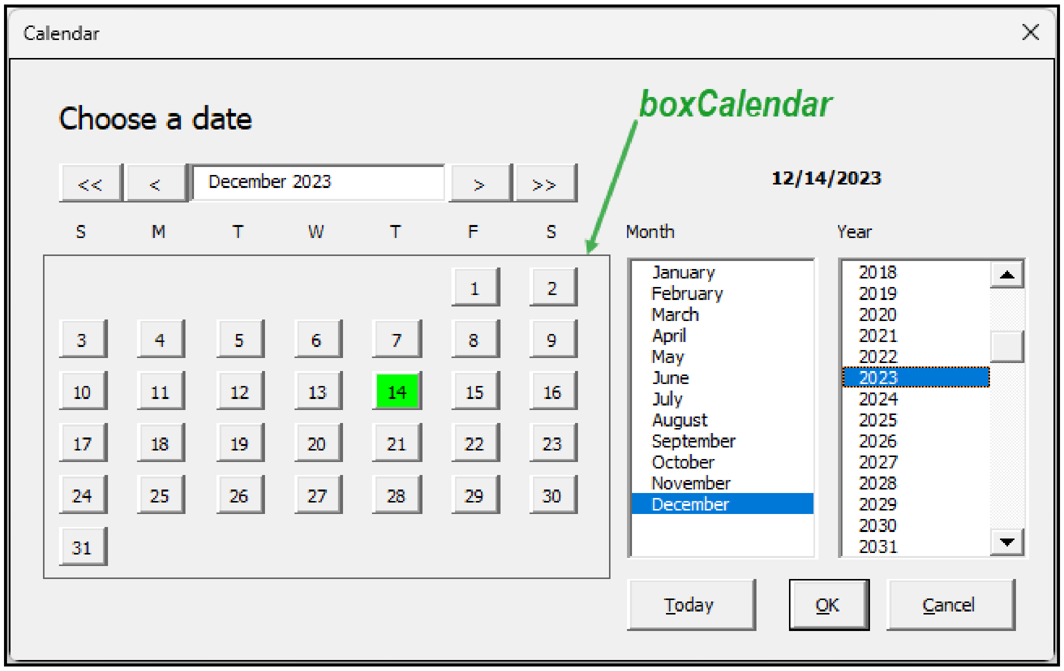


Figure 11: Form with a decorative box and its name shown in green.

After drawing a "box" around other objects, you should place the box behind the other controls, otherwise it will be difficult to select the controls that are in the box. To move it behind others, click on the edge of box then choose **Format** > **Order** > **Send to Back**.

When the label is used as a decorative line or box, feel free to use the prefix "lin" or "box" instead of "lbl" when naming the object.

C. How can I align objects horizontally and vertically?

You can align multiple objects together by doing the following:

1. Select one or more objects that need to be moved.

2. While holding down the control key, select the object to which the other object(s) will be aligned.

   You will notice that black squares appear on the object(s) that will be moved. You will also notice white squares appear on the object to which the others will be aligned.

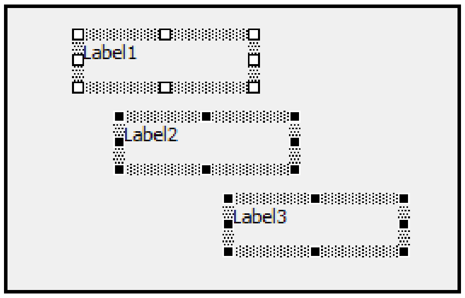


Figure 12: In this example, Label2 and Label3 will be aligned with Label1 because Label1 was selected last as indicated by its white squares.

3. Right click on one of the selected controls and choose **Align** or from the menu select **Format** > **Align**.

   As shown below you can then choose to align objects horizontally on their Left, Center, or Right side. Or you can also align objects vertically on their Top, Middle or Bottom side.

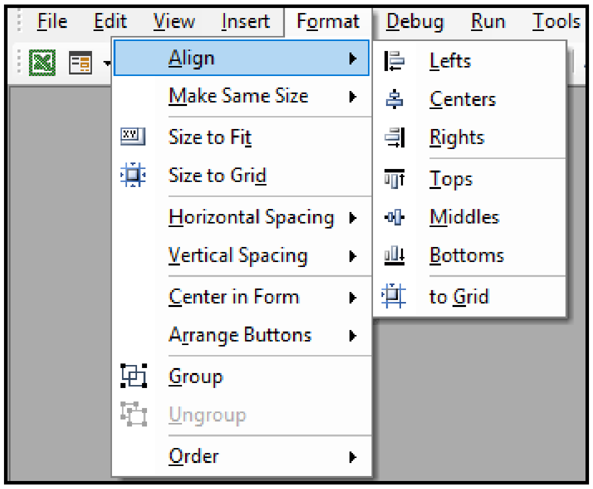
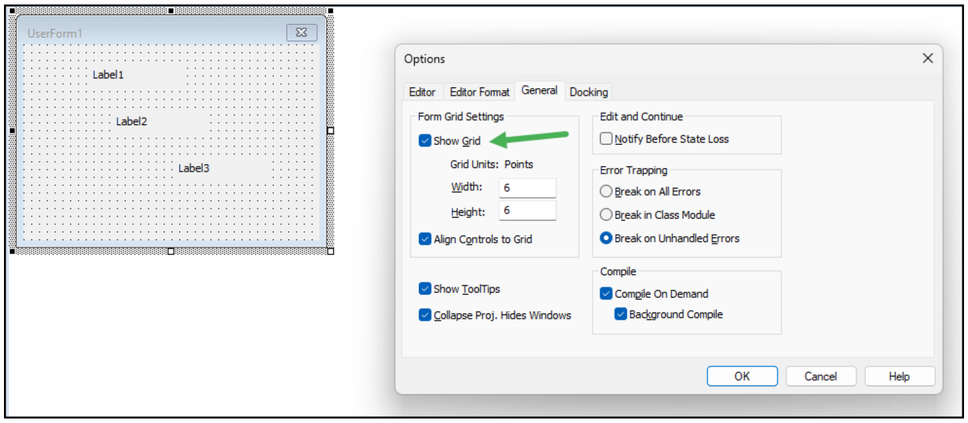


Figure 13

The final align menu option allows you to align objects to a grid. To view the grid, you should first select **Tools** > **Options** from the menu, click on the **General** tab, click the **Show Grid** option, and then click **OK**.



Form Grid Settings
- [✓] Show Grid
- Grid Units: Points
- Width: 6
- Height: 6
- [✓] Align Controls to Grid
- [✓] Show ToolTips
- [✓] Collapse Proj. Hides Windows

Edit and Continue
- [ ] Notify Before State Loss

Error Trapping
- ○ Break on All Errors
- ○ Break in Class Module
- ● Break on Unhandled Errors

Compile
- [✓] Compile On Demand
- [✓] Background Compile

Figure 14

Tip: An easy way to align a label and its text box

Aligning a label and a text box can be a little tricky because when they are aligned at the top, the words of the label do not appear to be aligned with the middle of the text box as shown in Figure 15 and Figure 16.

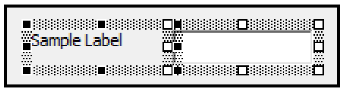


Figure 15

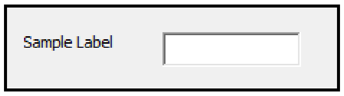


Figure 16

Instead of aligning them at the top, try this trick which came from a TechMentor student.

1. Set the label's **`AutoSize`** property to ***`True`***.
2. Select the label.
3. Control click on the text box.
4. From the menu choose **Format** > **Align** > **Middles**.

The label and text box will now be perfectly aligned as shown in Figure 17.

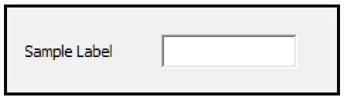


Figure 17

D. Exercise: Add labels and text boxes

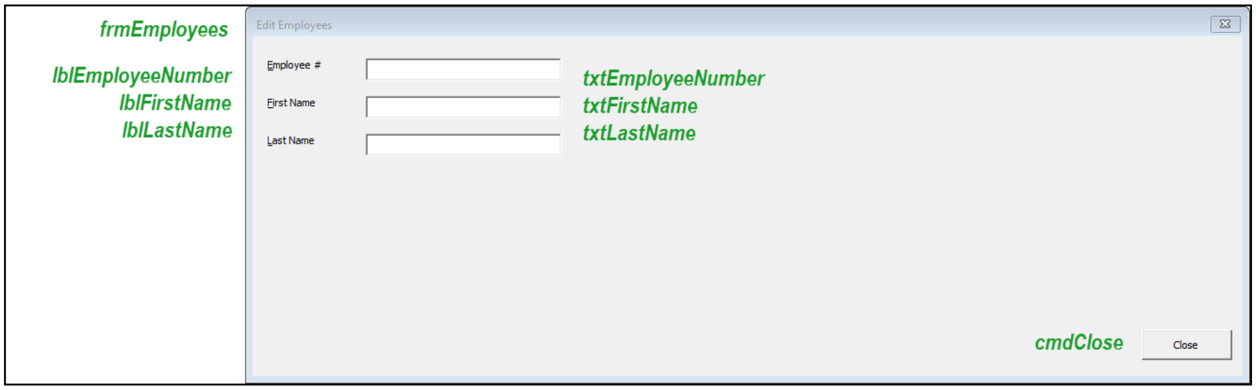


Figure 18

In this exercise you will resize the form you created earlier. You will then draw three pairs of labels and text boxes. You will give each object an appropriate name that uses the prefix shown in Table 1: Controls.

The order in which these objects are added will determine the default tab order for end users. Therefore, you will want to create the first label and then its text box before you create the second label and its text box, etc. You will learn how to modify the tab order in the next topic.

You will also apply a letter to the accelerator property to each label. This accelerator will allow users to jump from one control to another by simply pressing Alt and the accelerator letter.

1. View frmEmployees.
   If needed, in the Project explorer, click frmEmployees, and then click the View Object button.

2. In the properties window, set the following properties for the form.

Property	Value
Height	300
Width	800

3. Using the Toolbox's Select Objects button move the cmdClose button to the bottom far right of the form.

4. Using the Toolbox's **Label** button A and the text box button abl to draw three pairs of labels and text boxes as shown in Figure 18 above.

   Be sure to add each control in the order shown below and change their name, `Caption` and accelerator properties to the value shown below.

#	Object	(Name)	Caption	Accelerator
1	First label	lblEmployeeNumber	Employee #	E
2	First text box	txtEmployeeNumber		
3	Second label	lblFirstName	First Name	F
4	Second text box	txtFirstName		
5	Third label	lblLastName	Last Name	L
6	Third text box	txtLastName		

5. If desired, add an additional label as a decorative line or box of your choosing.

6. Press **F5** to run the form.

7. Press **Alt L**. The cursor should have moved to the Last Name box.

8. Press **Alt F**. The cursor should have moved to the First Name box.

9. Press **Alt E**. The cursor should have moved to the Employee # box.

10. Press **Tab**. The cursor should have moved to the first name box since it is the next box in the tab order.

11. Press **Shift Tab**. The cursor should have moved to the Employee # box since it is the previous box in the tab order.

12. Click the **Close** button.

13. Save the file.

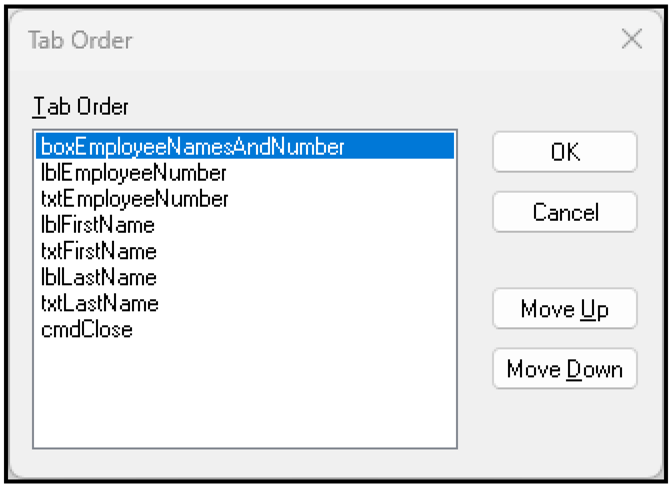
E. Tab Order

Many users prefer to use the tab key to move one control to the next. The order of those controls is the "tab order".

The default the tab order is determined by the order in which the controls are added to the form.

You can view and edit the tab order by selecting **View** > **Tab order** from the menu. As shown below, you can then select the object you want to move, and then click the Move Up or Move Down buttons until it is in the desired location.

A best practice is to place a label's tab order above its text box partner. This will ensure that the label's accelerator will cause the cursor to move to the text box since it cannot move to the label.



boxEmployeeNamesAndNumber
lblEmployeeNumber
txtEmployeeNumber
lblFirstName
txtFirstName
lblLastName
txtLastName
cmdClose

Figure 19

A control's tab order can also be set by changing its TabIndex property to a higher or lower value.

4. List boxes and Combo boxes

Complete this unit and you’ll be able to:

A. Describe the features and uses of a list box

B. Describe the features and uses of a comb box

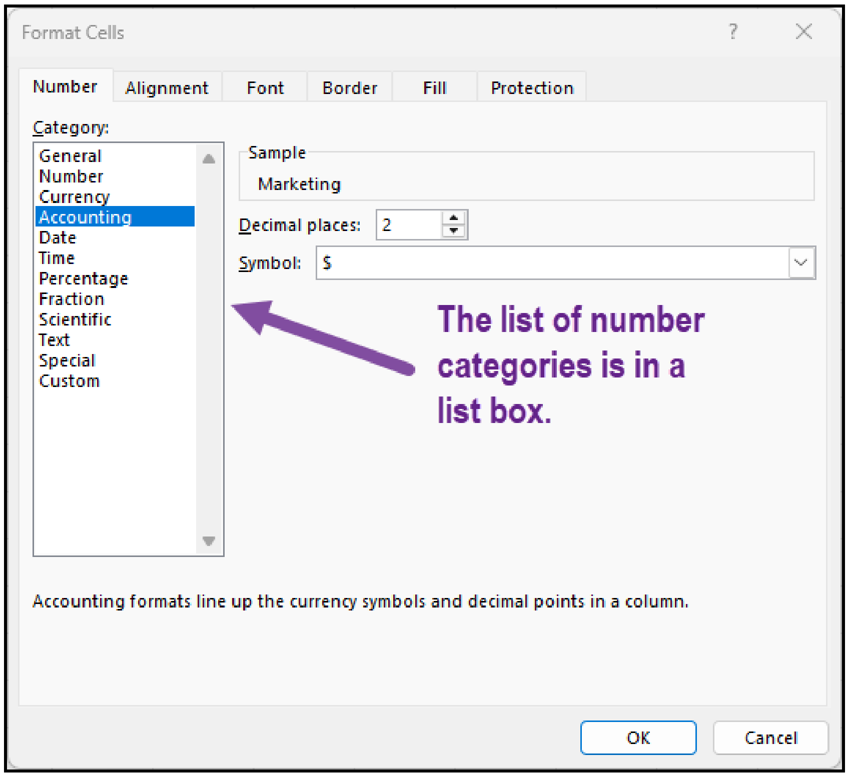
C. Populate a list box / combo box from a spreadsheet

D. Populate a list box / combo box via code

E. Exercise: Add a combo box to the employees form

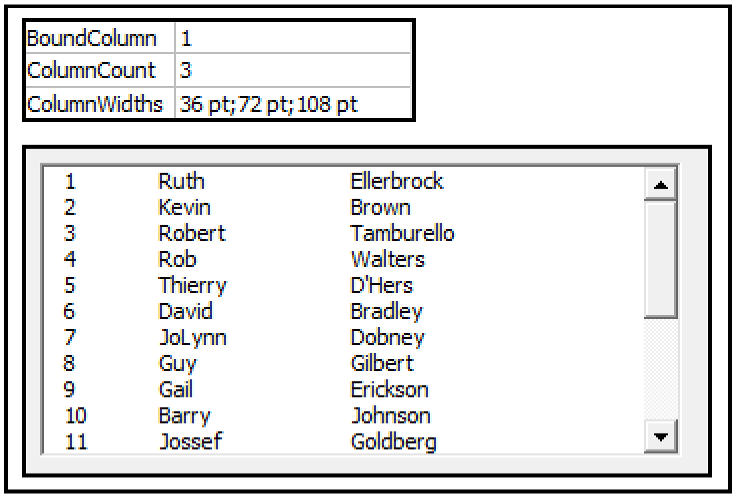
A. What is a List box?

A list box is a box with a list of items from which the user can select an item. For example, in Excel's Format Cells dialog box, the Category list is in a list box.



Accounting formats line up the currency symbols and decimal points in a column.

Figure 20



BoundColumn	1
ColumnCount	3
ColumnWidths	36 pt;72 pt;108 pt

1	Ruth	Ellerbrock
2	Kevin	Brown
3	Robert	Tamburello
4	Rob	Walters
5	Thierry	D'Hers
6	David	Bradley
7	JoLynn	Dobney
8	Guy	Gilbert
9	Gail	Erickson
10	Barry	Johnson
11	Jossef	Goldberg

Figure 21: A UserForm list box with 3 columns.

The list box has a `ColumnCount` property that determines the number of columns that appear in the list box. The `BoundColumn` property determines which column will control the value of the selected row.

The widths of each column can then be set with the **`ColumnWidths`** property. The list of column widths may be entered in inches or points (72 points is half an inch) and should be separated by semicolons as shown below. Note: If you enter the widths in inches, the computer will reset them as points.

Property	Value	Description
Column Count	3	The list box will have three columns.
Column Widths	0.5 in; 1 in; 1.5 in	The first column is half an inch wide, and the second and third are 1 inch and 1.5 inches wide. **Note: If you enter the widths in inches, the computer will reset them as points as shown below.**
Column Widths	36 pt;72 pt;108 pt	The first column is 36 points wide, and the second and third are 72 points and 108 points wide.

Tip: If you want to hide a column, set its width to ***`0 pt`***.

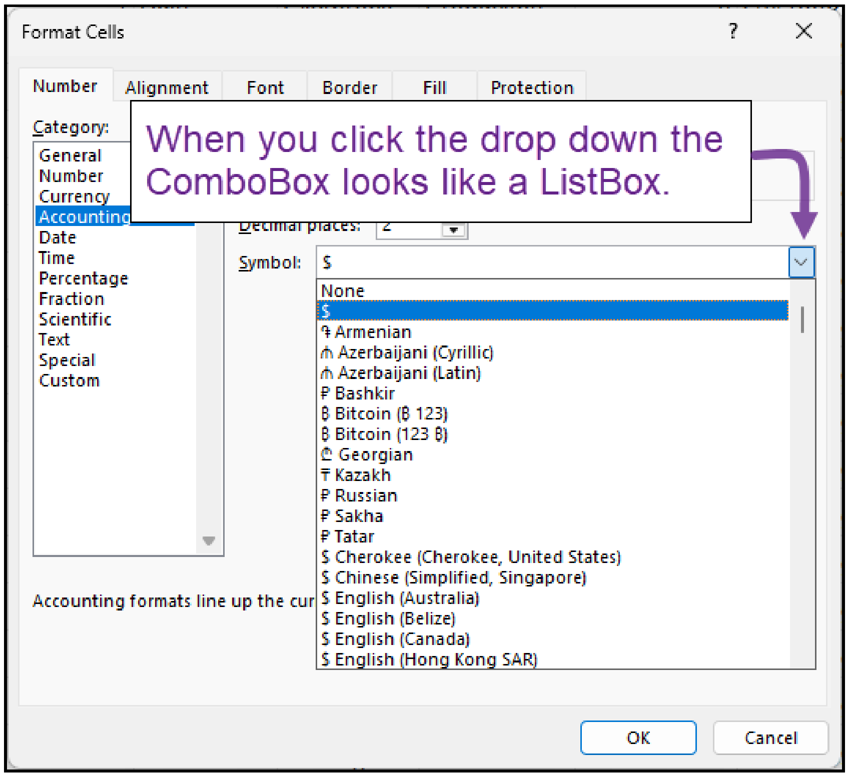
The list box can allow one or multiple selections based on its **`MultiSelect`** property's value.

Value	Description
`0 – frmMultiSelectSingle`	Allows the user to select only one value.
`1 - frmMultiSelectMulti`	User can click on each item they want selected.
`2 – frmMultiSelectExtended`	The user can select multiple items by clicking on the first and then shift-clicking on the last desired item.

After a user chooses an item from the list, its value will be stored in the **`Value`** property. This will usually be the value of the first column of the user's selected row. However, you can change that to a different column by entering the desired column's number in the **`BoundColumn`** property.

B. What is a Combo box

A combo box is like a list box, except it only displays one row. You can click its drop-down button to see the other items in the list. For example, in Excel's Format Cells dialog box, the Symbol list is in a combo box.



When you click the drop down the ComboBox looks like a ListBox.

General
Number
Currency
Accounting
Date
Time
Percentage
Fraction
Scientific
Text
Special
Custom

Decimal places: 2

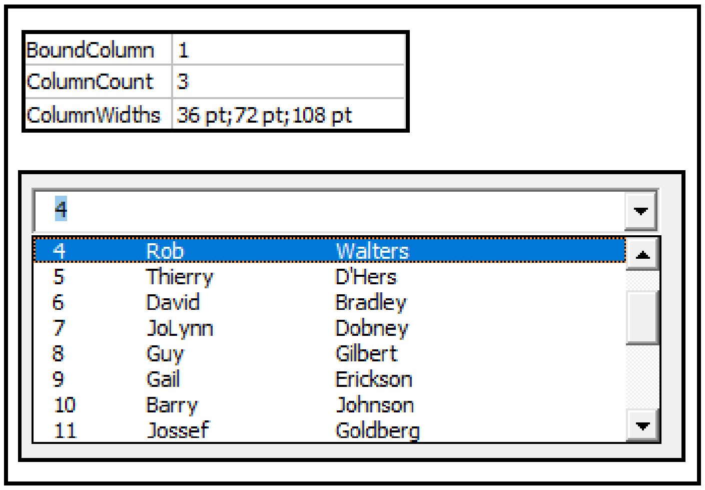
Symbol: $

None
$
֏ Armenian
₼ Azerbaijani (Cyrillic)
₼ Azerbaijani (Latin)
₽ Bashkir
₿ Bitcoin (₿ 123)
₿ Bitcoin (123 ₿)
₾ Georgian
₸ Kazakh
₽ Russian
₽ Sakha
₽ Tatar
$ Cherokee (Cherokee, United States)
$ Chinese (Simplified, Singapore)
$ English (Australia)
$ English (Belize)
$ English (Canada)
$ English (Hong Kong SAR)

Accounting formats line up the cur

Figure 22

You can determine the number of rows that appear when the user clicks the drop-down button by setting the **`ListRows`** property. Its default value is ***8***.



BoundColumn	1
ColumnCount	3
ColumnWidths	36 pt;72 pt;108 pt

4

4	Rob	Walters
5	Thierry	D'Hers
6	David	Bradley
7	JoLynn	Dobney
8	Guy	Gilbert
9	Gail	Erickson
10	Barry	Johnson
11	Jossef	Goldberg

Figure 23

As shown in Figure 23 the combo box can also have multiple columns via the `ColumnCount` and `ColumnWidths` properties. However, the additional columns will not be displayed until the user clicks the drop-down button.

Like the list box, the combo box also has a `Value` and a `BoundColumn` property. They work the same as in the list box.

C. How can I populate a list box or combo box via code?

One way to populate a list box or combo box is with code that is associated with the form's **Initialize** event. It uses the object's **AddItem** method and uses the desired value as the method's argument.

The following example code populates a combo box named cboGrade with a list of Elementary, Jr. High, and High School grades.

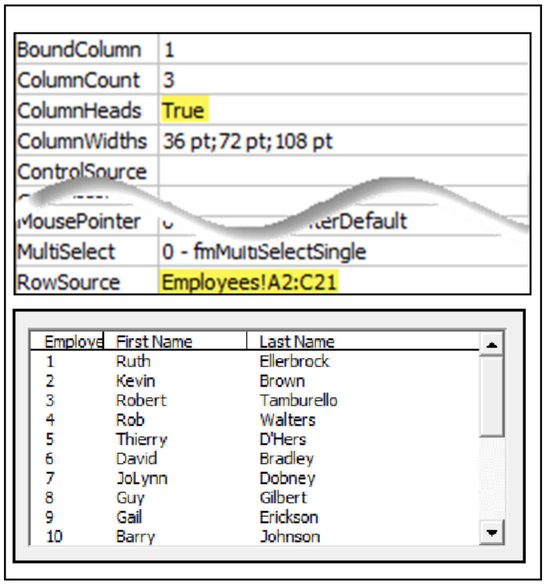
```vba
Private Sub UserForm_Initialize()
    Me.cboGrade.AddItem "Kindergarten "
    Me.cboGrade.AddItem "1st"
    Me.cboGrade.AddItem "2nd"
    Me.cboGrade.AddItem "3rd "
    Me.cboGrade.AddItem "4th"
    Me.cboGrade.AddItem "5th"
    Me.cboGrade.AddItem "6th"
    Me.cboGrade.AddItem "7th"
    Me.cboGrade.AddItem "8th"
    Me.cboGrade.AddItem "9th Freshmen"
    Me.cboGrade.AddItem "10th Sophomore"
    Me.cboGrade.AddItem "11th Junior"
    Me.cboGrade.AddItem "12th Senior"
End Sub
```

D. How can I populate a list box or combo box from a spreadsheet?

You can populate your list box and/or combo box by setting the **RowSource** property to a range of cells as shown below. If your workbook has multiple sheets, your **RowSource** should include the sheet name, an exclamation mark, and then the range of cells. You can also set the **RowSource** to a NamedRange.

Property	Value	Description
RowSource	A2:C21	Range refers to cells A2 through C21 of the active sheet.
RowSource	Employees! A2:C21	Range refers to cells A2 through C21 of the Employees sheet. See Figure 24.
RowSource	Departments	Assuming that "Departments" is a named range, the combo box or list box will refer to the cells that the named range refers to.

If you set the **ColumnHeads** property to ***True*** then the computer will retrieve column headings from the range of cells above your row source. For example, the values in A1:C1 would be the column headings if the row source refers to A2:C21.



BoundColumn	1
ColumnCount	3
ColumnHeads	True
ColumnWidths	36 pt;72 pt;108 pt
ControlSource	
MousePointer	...erDefault
MultiSelect	0 - fmMultiSelectSingle
RowSource	Employees!A2:C21

Employe	First Name	Last Name
1	Ruth	Ellerbrock
2	Kevin	Brown
3	Robert	Tamburello
4	Rob	Walters
5	Thierry	D'Hers
6	David	Bradley
7	JoLynn	Dobney
8	Guy	Gilbert
9	Gail	Erickson
10	Barry	Johnson

Figure 24

E. Exercise: Add a combo box for Departments

In this exercise you will first create a supporting list sheet, and add a list of departments to that sheet. You will then create a Dynamic Named Range with a formula that will ensure if you add Departments, they will be included in the named range. Finally, you will add the combo box that will get its `RowSource` from the Named Range.

1. If needed, open your Employees.xlsxm file in Excel.

2. Add a new sheet.

3. Name the new sheet "**Supporting Lists**".

4. In cell A1, type "Departments", and make it bold.

5. Enter the following Departments below A1.
   - Accounting
   - Finance
   - Human Resources
   - Sales

6. AutoFit column A's width.

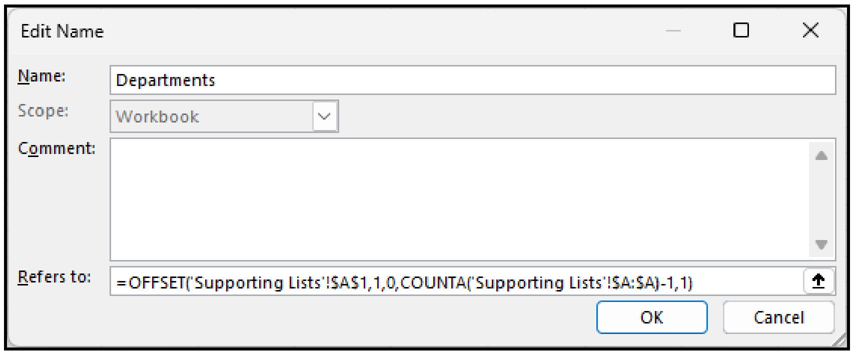
7. Click on cell A1.

8. From the ribbon select **Formulas** > **Name Manager** and click the **New…** button.

9. In the New Name dialog box, verify the **Name** is "**Departments**"

10. In the **Refers to:** area enter the following formula:

    =OFFSET('Supporting Lists'!$A$1,1,0, COUNTA('Supporting Lists'!$A:$A)-1,1)

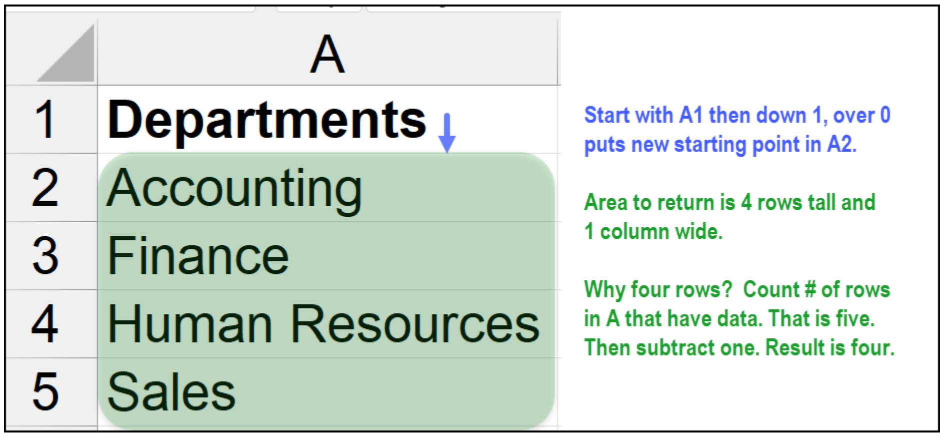


Name: Departments
Scope: Workbook
Comment:
Refers to: =OFFSET('Supporting Lists'!$A$1,1,0,COUNTA('Supporting Lists'!$A:$A)-1,1)

Figure 25

This formula uses the OFFSET function. The syntax for the OFFSET function is:
OFFSET(reference, rows, cols, [height], [width])

Argument	Description	Value from our formula
reference	The starting cell or range from which the offset begins.	A1
rows	The number of rows down from the starting reference (negative for up).	1 Reference is now A2.
cols	The number of columns to the right from the starting reference (negative for left).	0 Reference is still A2.
height	(Optional) Height (rows) to return from the calculated reference. Default is 1.	4 Count rows in A then sub-tract 1.
width	(Optional) Width (columns) to return from the calculated reference. Default is 1.	1


A
1 Departments
2 Accounting
3 Finance
4 Human Resources
5 Sales
Start with A1 then down 1, over 0 puts new starting point in A2.
Area to return is 4 rows tall and 1 column wide.
Why four rows? Count # of rows in A that have data. That is five. Then subtract one. Result is four.

11. Click **OK**.

12. Click **Close**.

13. Test the named range by pressing **Ctrl G**, and typing **Departments** and clicking **OK**. The four departments should be highlighted

14. Press Alt F11 to open the VB Edit.

15. If needed double click on the form.

16. Add a new label under the `lblLastName` label.

17. Set the `(Name)` of the new label `lblDepartment` and set the `Caption` to `Department`.

18. Add a combo box control to the right of the department label.
Set its `(Name)` to `cboDept`. Set its `RowSource` to `Departments` which is the name of the NamedRange you created in steps 9 through 12.

19. Press **Ctrl S** to save the workbook and its form.

20. Press **F5** to test the form.

21. Click the Department drop down and verify it has the four departments from the supporting lists sheet.

22. Click the **Close** button.

23. Return to the Supporting Lists sheet and add “Production” to cell A6.

24. Go test the form again. The combo box should now include five departments including “Production”.

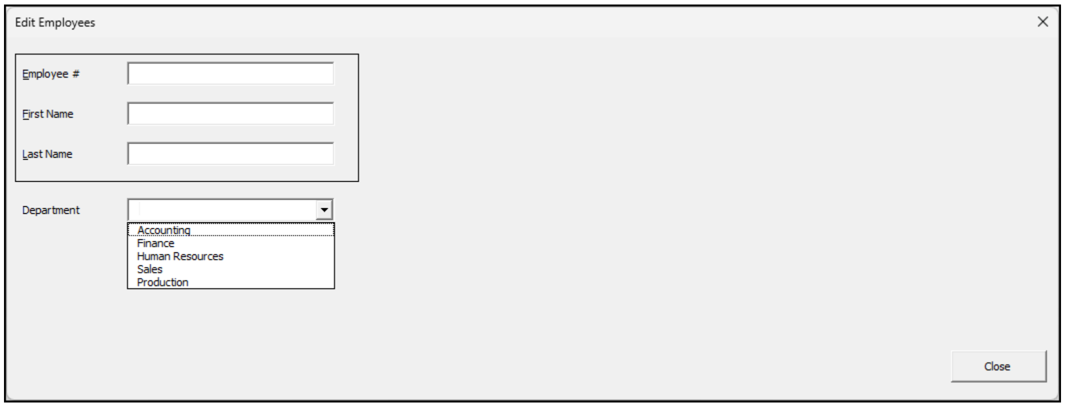


Figure 26

25. Click the **Close** button.

26. Save your workbook.

5. Spin buttons and Scroll bars

Complete this unit and you'll be able to:

A. Describe the features and uses of a spin button

B. Describe the features and uses of a scroll bar

C. Exercise: Add a spin button to your UserForm

A. What is a Spin button?

A spin button is a control that allows users to increase or decrease a value by clicking an up or down arrow. It has a `Value` property which must be a positive whole number. By default, it is 0.

The value of the spin button does not appear on the UserForm. Instead, you, the designer, are expected to add a partner control, such as a text box, that will show the spin button's value. You will also need to write VBA code to keep the text box and the spin button in synch with each other. You will see how this is done in C. Exercise: Add a spin button to your UserForm.

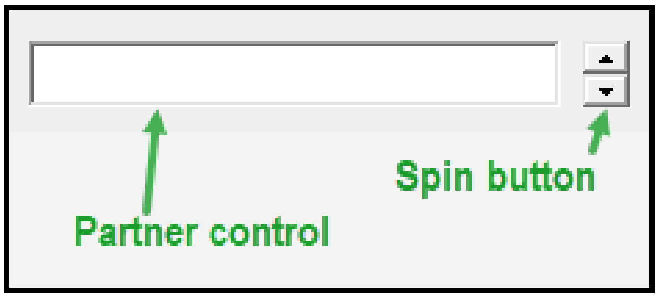


Figure 27

The spin button has `Min` and `Max` properties that set minimum and maximum values for the spin button.

By default, when you click the up or down arrow, the spin button value increases or decreases by one. You can, however, set the `SmallChange` property to a different number value if you want to increase / decrease the value at a higher rate.

When a user clicks the up or down arrow, the **Change** event will execute, and so will either the **SpinUp** or **SpinDown** event. If the user clicks and holds down the up or down arrow, the events will occur repeatedly. Therefore, you the designer can change the `Delay` property to set the number of milliseconds that occur between each execution. For more information, see the following web page: https://learn.microsoft.com/en-us/office/vba/Language/Reference/User-Interface-Help/delay-property.

B. What is a scroll bar?

In Excel, there are scroll bars that lets you quickly scroll to columns or rows that are off the screen. That is NOT what the scroll bar control does.

The scroll bar control is like an advanced a spin button; it lets you increase or decrease a value that is shown in a related control. It is advanced because the `LargeChange` propery gives the user the option to make a large change by clicking within the scroll bar, and the `SmallChange` property lets users make small change by clicking the arrows. Users can also make changes by sliding the "scroll box" that is within the scroll bar's track.

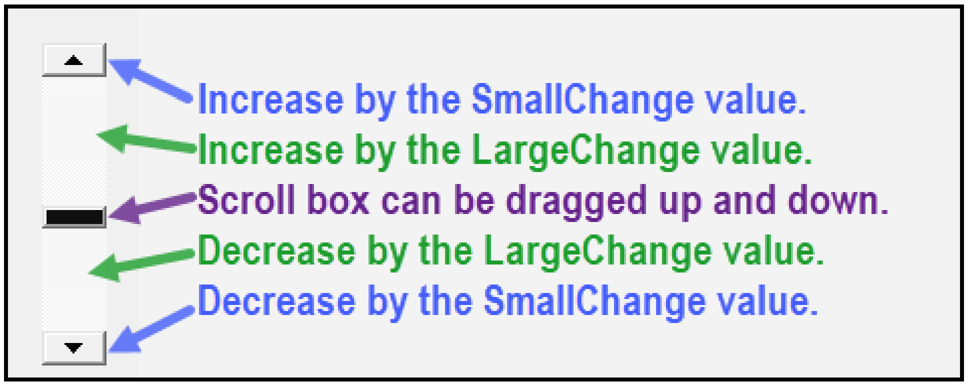

Increase by the SmallChange value.
Increase by the LargeChange value.
Scroll box can be dragged up and down.
Decrease by the LargeChange value.
Decrease by the SmallChange value.

Figure 28

C. Exercise: Add a spin button to your UserForm

1. Open your Employees – WIP.xlsm file in Excel.
2. Go to the Visual Basic Editor and display the frmEmployees form.
3. Add a label named ***lblDependents***, and a text box named ***txtDependents***.
4. Set the label's **Caption** to ***Dependents***.
5. Set the text box's **value** to ***0***.
6. From the toolbox click on the Spin button control, then click on the form, just to the right of ***txtDependents***.
7. Set the height equal to the height of ***txtDependents***.
8. Name the Spin button ***spnDependents***.
9. Set the spin button's **max** property to ***30***.
10. Double click on the spin button to go to the code window.
11. Change the code for the **spnDependents_Change** event to the following:

```
Private Sub spnDependents_Change()
    'Set the value of the text box to the value of the spin button.
    Me.txtDependents.Value = Me.spnDependents.Value
End Sub
```

Code_block 1

12. From the procedure drop down menu select **txtDependents**, and if needed choose **Change** from the property drop-down menu.
13. Change the code for the txtDependents_Change event to the following:

```
Private Sub txtDependents_Change()
    'Create an integer variable to track the number of dependents.
    Dim intDependents As Integer

    'If the new value is a number then...
    If IsNumeric(Me.txtDependents.Value) Then

        'Convert the value to an integer, and store it in the variable.
        intDependents = CInt(Me.txtDependents.Value)

        'If the value is between the min and max then...
        If intDependents >= Me.spnDependents.Min _
        And intDependents <= Me.spnDependents.Max Then
            'Set the spnDependents.value to that number.
            Me.spnDependents.Value = intDependents
        Else
            'Otherwise reset the text to the value of the spin button.
            Me.txtDependents.Value = Me.spnDependents
        End If
    Else
        'Otherwise reset the text to the value of the spin button.
        Me.txtDependents.Value = Me.spnDependents
    End If
End Sub
```

Code_block 2

14. Press **F5** to test the form and answer the following questions:

- Can you type numbers in the dependents text box?
  Hopefully yes.
- Can you type numbers greater than 30?
  Hopefully not.
- Can you enter a negative number?
  Hopefully not.
- Can you enter text?
  Hopefully not.
- When you click the up or down arrows, do the number of dependents in the text box increase or decrease by one?
  Hopefully yes as long as the numbers are between 0 and 30.

6. Working with multiple UserForms

Complete this unit and you'll be able to:

A. Duplicate UserForms

B. Export a UserForm to a file

C. Import a UserForm from a file

D. Remove a UserForm from an Excel workbook

E. Exercise: Import a UserForm Calendar

F. Add code to work with multiple forms

G. Exercise: Add controls and code to work with the calendar form

A. How do I duplicate a UserForm?

Sometimes you will want to copy and paste a UserForm from one Excel workbook into another. Unfortunately, copying and pasting will not work. But, dragging and dropping will. Proceed as follows:

1. Open the source workbook (the workbook that has the UserForm you want to copy).

2. Open the destination workbook (the workbook that will receive the UserForm).

3. Switch to the Visual Basic Editor.

4. Verify the Project Explorer is open, and that you can see both workbooks and their forms.

5. Drag and drop the desired UserForm from the source workbook to the destination workbook.

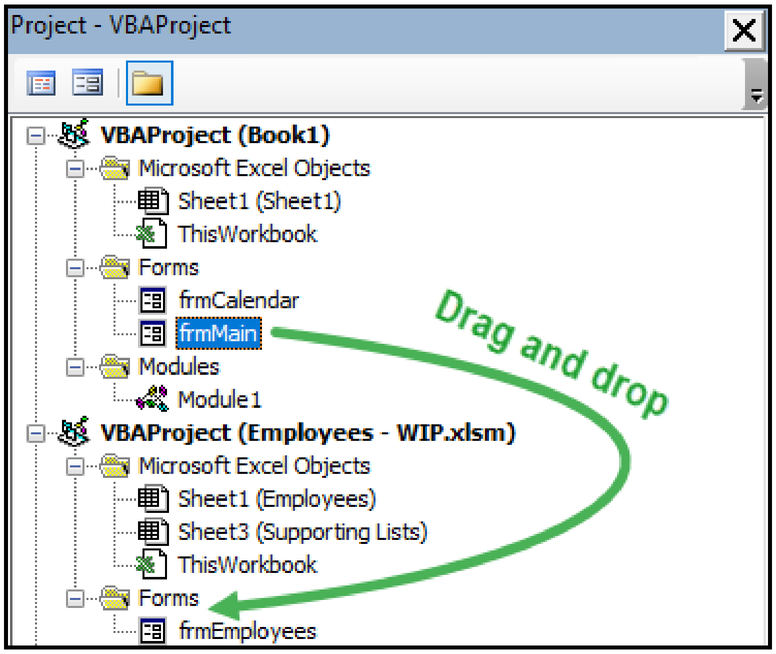


Figure 29

6. Save your destination workbook.

B. How can I export a UserForm to a file?

Another way to duplicate a UserForm between workbooks is to export the UserForm from the source workbook file and then import it into the destination workbook. In this topic you will see how to export it. In the next topic you will learn how to import it.

Do the following to export a UserForm:

1. Open the source workbook.
2. Switch to the Visual Basic Editor.
3. Right click on the form you want to export and choose **Export File…**

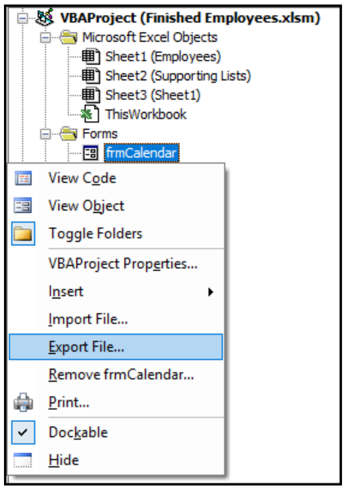
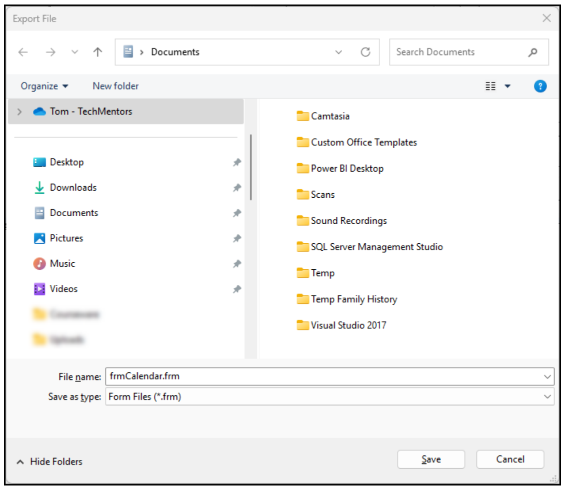


Figure 30

4. When prompted, choose a folder, enter a filename, and click **Save**. Notice the file will have a .frm extension.



File name: frmCalendar.frm
Save as type: Form Files (*.frm)

Figure 31

C. How can I import a UserForm from a file?

Importing a UserForm from a file is fairly easy. Proceed as follows:

1. Open the destination workbook.
2. Switch to the Visual Basic Editor.
3. Right click on the destination workbook and choose **Import File…**

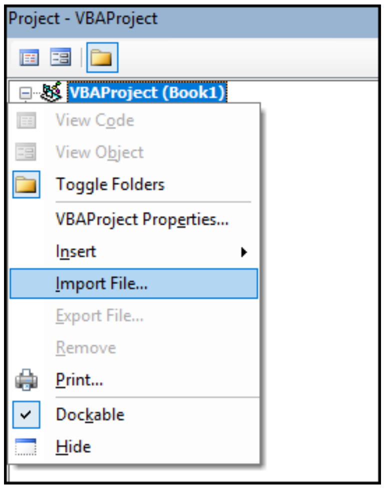


Figure 32

4. Select the desired .frm file from its folder and choose **Open**.
5. **Save** your destination workbook.

D. How can I remove a UserForm from an Excel workbook?

You can easily remove a UserForm from your workbook by doing the following:

1. Open your workbook in Excel.

2. View the Visual Basic Editor, and verify you can view the form.

3. Right click on the form and choose **Remove *formname*…**.

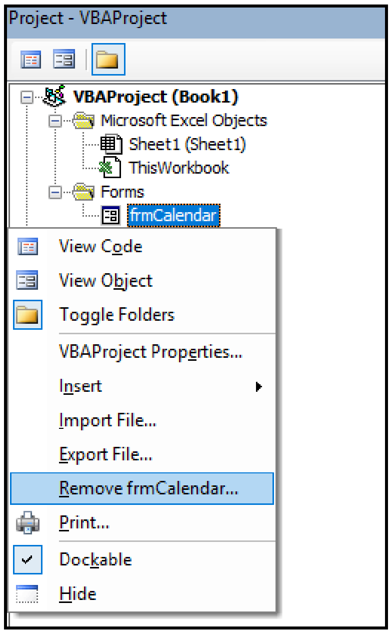
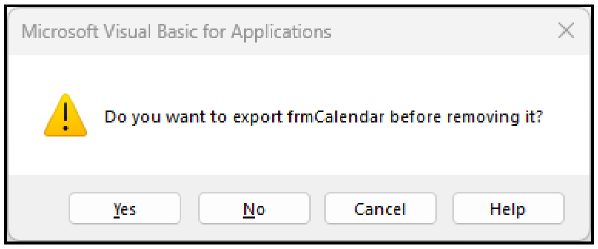


Figure 33

4. Respond to the message box that appears asking if you want to export the form before removing it. If you do not need the form exported, click **No**.



Microsoft Visual Basic for Applications

Do you want to export frmCalendar before removing it?

Figure 34

The form will then be removed from your workbook.

5. Save your workbook.

E. Exercise: Import a UserForm Calendar

In this exercise you will import a UserForm Calendar from a file named **frmCalendar.frm**.

1. Start Excel and open your **Employees – WIP.xlsm** workbook.

2. Switch to the Visual Basic Editor.

3. Right click on the Employees – WIPD.xlsxm workbook and choose **Import File…**

4. Select the **frmCalendar.frm** file from the Class_Files folder and choose **Open**.

5. Double click the **frmCalendar** form to verify it imported properly.

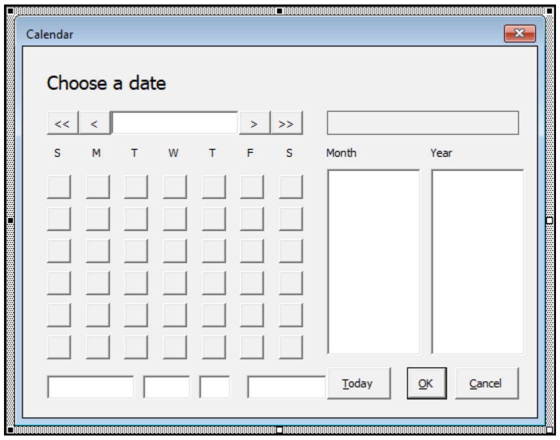


Figure 35

6. Press **F5** to test the form.

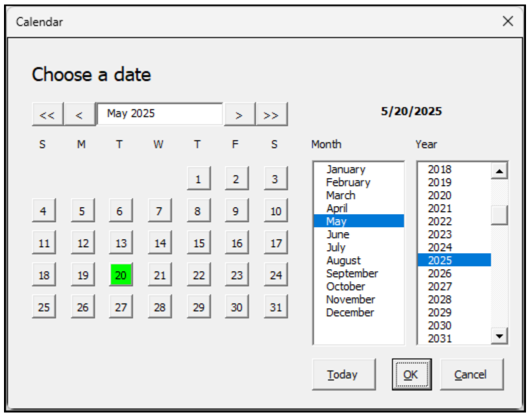


Figure 36

7. Play with the various drop downs, buttons, etc. to verify it works as expected.

8. Click the OK or the Canel button to close the form.

9. **Save** your **Employees – WIP.xlsm** workbook.

F. How can I add code to work with multiple forms

Working with multiple forms can quickly get complicated. For example, in our employee form we want to add a Hire Date label, text box, and button. The label of course will say "Hire Date" and the text box will obviously show the date the employee was hired. The button will need to be programmed to open frmCalendar so the user pick the hire date. The user will then click OK from the calendar form, which will make the calendar disappear. And if it has disappeared, then how to do find out which date they picked? The answer is with VBA procedures.

VBA Procedures

VBA includes multiple types of procedures including sub procedures for macros, event procedures for our forms, and function procedures that calculate results.

User-defined Functions

A user-defined function's declaration statement has the following five parts:

Public or Private	This identifies if the function can be called from anywhere (Public) or if it can only be called from within the same module (Private).
Function	This keyword identifies that it is a function procedure.
Name	Identifies the name of the function. The name must start with a letter and have no spaces.
Parenthesis and arguments	The set of parenthesis is required. Inside the parenthesis you have the option to list variables (arguments) that will receive values from outside the function.
Assignment	This starts with the keyword "As" and then identifies the type of data that the function will return. For example "As String" will return a string of text, while As Integer will return an integer value.

Consider the following function procedures that we will put into a new module in the next exercise.

IsLoaded()

This function is used to determine if a specific UserForm is open. The name of the UserForm is passed into the function's FormName argument. The code then loops through the list of open forms, and checks if the opened form's name matches the one we are looking for. If it matches, then the function returns True. After looping through all the open forms, if our form is not found then function returns False.

```
Public Function IsLoaded(FormName As String) As Boolean
    ' Purpose:  Determines if a form is loaded (opened).

    'Create a variable to track user forms.
    Dim obj As Object

    'Loop through each form object in the collection of open forms to see if our
desired form is open.
    For Each obj In VBA.UserForms

        'If the object's name matches our FormName argument, then return True and exit
the function.
        If StrComp(obj.Name, FormName, vbTextCompare) = 0 Then
            IsLoaded = True
            Exit Function
        End If

    Next obj

    'If the code gets this far it means our form was not open, therefore return False
    IsLoaded = False
End Function
```

Code_block 3

GetDate()

This function is used to open the calendar form and get a date from the user. If the user clicked on a date and clicked OK, then this function will return that date. If they clicked cancel, this function will return the date we initially passed into this function.

```
Public Function GetDate(ExistingDate As String, Description As String) As String
    'Purpose: Gets a date from the user by letting them pick it from the frmCalendar.

    'Load frmCalendar, but initially keep it hidden from the user
    Load frmCalendar

    With frmCalendar
        'If we passed a value to the ExistingDate argument then...
        If Len(ExistingDate) Then
            'Set the calendar equal to that date.
            .Calendar1 = ExistingDate
        End If

        'Change the Description label's caption to match the value of the Description
we passed in.
        .lblDescription.Caption = " " & Description

        'Now show the calendar form.
        .Show
    End With

    'While the form is displayed on the screen, the code below will not run.
    'Once the form is closed or hidden, the code below will run.
    'If the form is loaded then...
    If IsLoaded("frmCalendar") Then

        'Set our function equal to the date the user chose.
        GetDate = Format(frmCalendar.txtChosenDate, "mm/dd/yyyy")

        'Now close the form.
        Unload frmCalendar
    Else
        'If the form is not opened (meaning the user clicked cancel) then ...

        'Set our function equal to the value we passed into the ExistingDate argument.
        GetDate = ExistingDate
    End If
End Function
```

Code_block 4

ShowForm()

This macro will show the frmEmployees.

```
Public Sub ShowForm()
    frmEmployees.Show
End Sub
```

Code_block 5

G. Exercise: Add controls and code to work with the calendar form

In this exercise, you will create a module for the code we just reviewed. You will then to the employee form a Hire Date label, Hire Date text box, and a button that will let the user pick a Hire Date from the Calendar form.

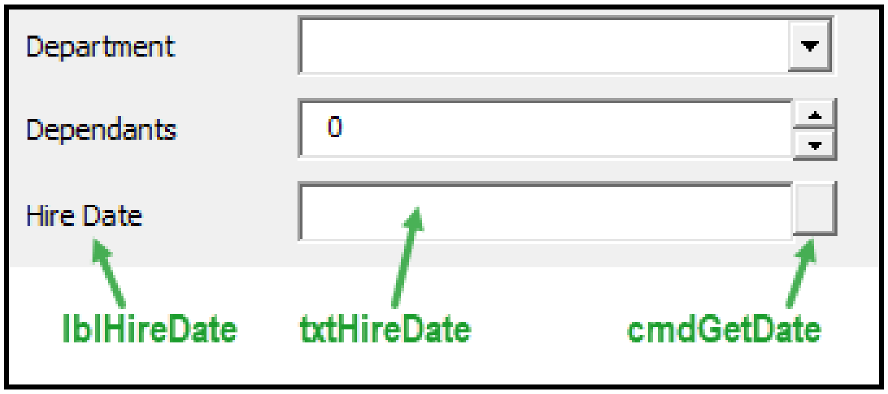
1. If needed Start Excel and open your **Employees – WIP.xlsm** workbook.

2. Switch to the Visual Basic Editor and display **frmEmployees**.

3. Add a label named ***lblHireDate*** below the Dependents label. Set its **Caption** to ***Hire Date***.

4. Add a text box named ***txtHireDate*** below the Dependents text box.

5. Add a command button named ***cmdGetDate*** below the Dependents spin button. Make it the same width as the width of the spin button. Erase its **Caption**.



Code_block 6

6. From the menu choose **Insert** > **Module**.

7. Add the code from F. How can I add code to work with multiple forms into the module.

8. Go to the frmEmployees, double click the Get Date command button, and enter the following code.

```
Private Sub cmdGetDate_Click()
    'Create a variant variable to track the date.
    'A variant is a variable that can have any type of value,
text, number, date, etc.
    Dim varDate As Variant

    'If the length of the Hire Date text box value is 0 (ie has
no text) then...
    If Len(Me.txtHireDate) = 0 Then
        'Get the date from the user by running the GetDate
function. Use the current date as the default date.
        varDate = GetDate(ExistingDate:=Date, Description:="Hire
date")
    Else
        'Get the date from the user by running the GetDate
function. Use the value of txtHireDate as the default date.
        varDate = GetDate(ExistingDate:=Me.txtHireDate,
Description:="Hire date")
    End If

    'If the user picked a date, then ...
    If Len(varDate) Then
        'Set the txtHireDate to that date formatted as Month
day, year.
        Me.txtHireDate = Format(varDate, "mmmm d, yyyy")
    End If
End Sub
```

Code_block 7

9. Go to Excel and run the ShowForm() macro.

10. Click the Get Hire Date button, pick a date and click **OK**.
The hire date text box should be filled in with your selected date.

11. Click the **Close** button, and save your workbook.

7. Check boxes and option buttons

Complete this unit and you'll be able to:

A. Describe the features and uses of a check box

B. Describe the features and uses of option buttons

C. Describe the features and uses of a frame control

D. Exercise: Add a check box to your UserForm

E. Exercise: Add a group of option buttons to your UserForm

A. What is a check box?

As shown below, a check box is a control that allows the user to make a yes or no, on or off, true or false choice by checking or unchecking a small square box. The control has a **`Caption`** which appears to the right of the small square box. It also has a **`value`** property that is either ***`True`*** or ***`False`***. The default value is ***`False`***.

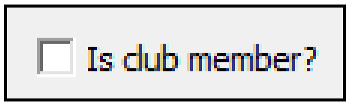


Figure 37

B. What is an option button?

An option button is a circle

An option button is used when you want to let the user pick one of a series of options. Each option button displays a small circle that can be selected or unselected. Next to the circle is the option button's **`Caption`**.

Multiple option buttons can be grouped together via a common **`GroupName`**. For example, in Figure 38 the four grade-related option buttons all have a **`GroupName`** of ***`Grade`*** and the three gender-related option buttons have a **`GroupName`** of ***`Gender`***.

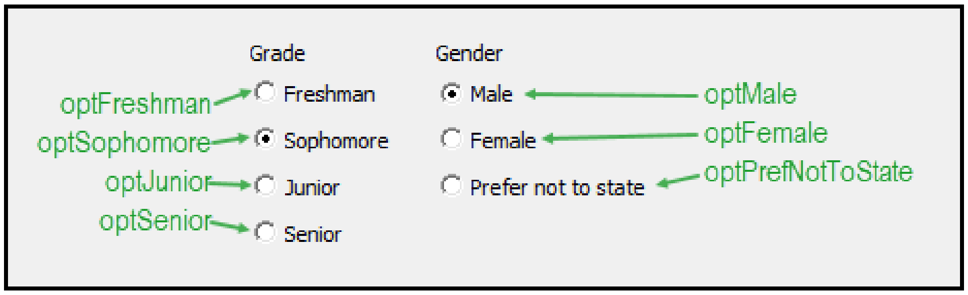


Figure 38

Each option must have its own unique **`(Name)`**. As shown above, a best practice is to use the "opt" prefix plus a descriptive name that matches the **`Caption`**.

Each option button has a **`value`** property that is either ***`True`*** or ***`False`***. The default value is ***`False`***. To find out which option was chosen, your code can use a series of `If… ElseIf` statements like the shown below.

```
Dim strGender As String

If optMale.Value = True Then
    strGender = "Male"
ElseIf optFemale.Value = True Then
    strGender = "Female"
ElseIf optPreferNot.Value = True Then
    strGender = "Prefer not to state"
Else
    strGender = "No selection"
End If
```

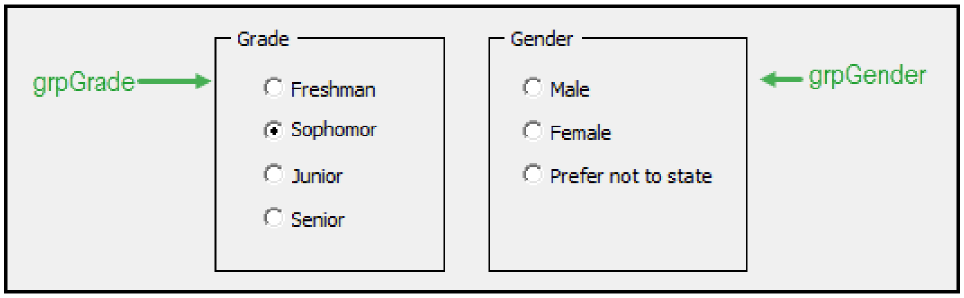
Code_block 8

C. What is a frame control?

A frame control is a container that groups related controls together. Labels, text boxes, option buttons, check boxes, and other controls can all be put in the same frame container. This allows you to visually group like controls together. It also allows you to easily move all the grouped controls together by simply moving the frame.

The **(Name)** of the container should start with the prefix ***grp*** (meaning Group) because the prefix ***frm*** refers to a form.

The frame has a **Caption** property. Its value will appear at the top left of the container.



Code_block 9

Controls placed in the frame are considered children of the frame. If you move the frame, its children will move with it. Also, children of the frame will not appear in the **Properties Window drop-down**, nor the **View** > **Tab Order** dialog box unless you first select the frame.

Controls placed in a container are also logically grouped together. Therefore, when multiple option buttons are put in same the frame they do not need to have a common **GroupName**.

By default, the frame will have a border surrounding it. You can make this invisible by changing the **BorderStyle** property to ***0 – frmBorderStyleNone***.

D. Exercise: Add a check box to your UserForm.

In this exercise you will add an Is Temporary check box to your Employees user form to identify if the employee is permanent or temporary.

1. If needed Start Excel and open your **Employees – WIP.xlsm** workbook.
2. Switch to the Visual Basic Editor and display **frmEmployees**.
3. Add a check box named ***`chkIsTemp`*** below and to the right of the Hire Date controls.
4. Set the **`Caption`** to ***`Is Temporary`***.
5. Save your workbook.

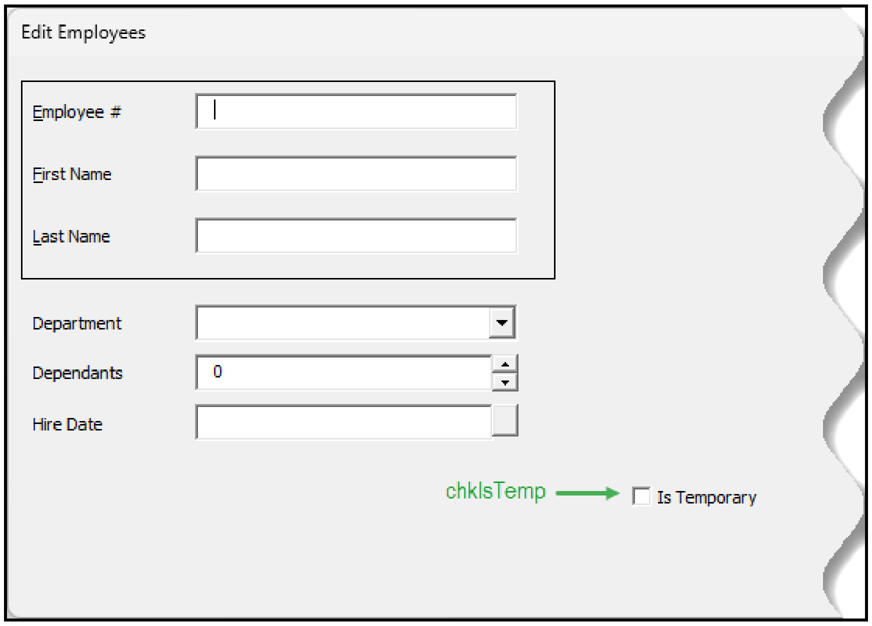


Figure 39

E. Exercise: Add a group of option buttons to your UserForm.

1. If needed Start Excel and open your **Employees – WIP.xlsm** workbook.
2. Switch to the Visual Basic Editor and display **frmEmployees**.
3. Add a frame named ***`grpRegion`*** to the right of employee number text box.
4. Set the **`Caption`** to ***`Region`***.
5. Add the following option buttons to the frame and give them the **`(Name)`** and **`Caption`** specified.

`(Name)`	**`Caption`**
optCanada	Canada
optMexico	Mexico
optNortheast	Northeast
optNorthwest	Northwest
optSoutheast	Southeast
optSouthwest	Southwest

8. Loading and Saving UserForm Data in Excel

Complete this unit and you'll be able to:

A. Write code to populate your UserForm from Excel data

B. Exercise: Populate your UserForm with data from Excel

C. Write code to save UserForm data to Excel

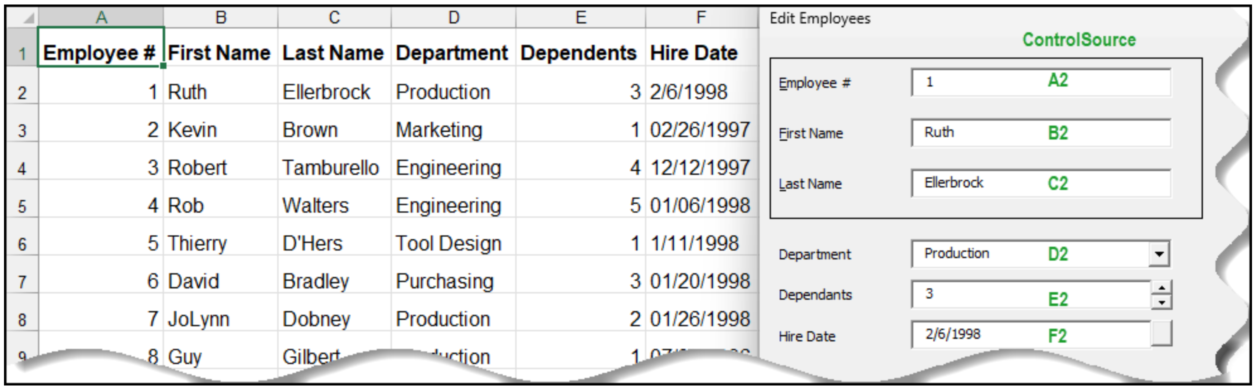
D. Exercise: Save your UserForm data to Excel

A. How can I populate my UserForm from my Excel worksheet?

You can populate the controls on your UserForm either by using the `ControlSource` property or by writing VBA code.

ControlSource method

Text boxes, list boxes, combo boxes, spin buttons, check boxes, option buttons and other controls have a `ControlSource` property that can be set equal to a cell in Excel. For example, Figure 40 below shows the Employee sheet and the Employee form where the `ControlSource` property has been set to the cell reference listed in green.



	A	B	C	D	E	F
1	Employee #	First Name	Last Name	Department	Dependents	Hire Date
2	1	Ruth	Ellerbrock	Production	3	2/6/1998
3	2	Kevin	Brown	Marketing	1	02/26/1997
4	3	Robert	Tamburello	Engineering	4	12/12/1997
5	4	Rob	Walters	Engineering	5	01/06/1998
6	5	Thierry	D'Hers	Tool Design	1	1/11/1998
7	6	David	Bradley	Purchasing	3	01/20/1998
8	7	JoLynn	Dobney	Production	2	01/26/1998
9	8	Guy	Gilbert	[illegible]	1	[illegible]

Edit Employees

		ControlSource
Employee #	1	A2
First Name	Ruth	B2
Last Name	Ellerbrock	C2
Department	Production	D2
Dependants	3	E2
Hire Date	2/6/1998	F2

Figure 40: Employees sheet and form with the value of the `ControlSource` property shown in green.

This method works well if the location of the data for each control never changes. But when the location of the data changes, using the control source does not work. For example, if you want to edit data for employee #2, then the control sources would need to change to A3, B3, C3, D3, E3, and F3.

VBA Code method

With the VBA code method, you write code that updates the control's value. Each statement will need to include the name of the control to populate, an equal sign, and then a reference to the workbook, worksheet, and the desired cell address.

For example, the code below will populate the form for employee #1.

```
Me.txtEmployeeNumber = ThisWorkbook.Sheets("Employees").Cells(2, "A")
Me.txtFirstName = ThisWorkbook.Sheets("Employees").Cells(2, "B")
Me.txtLastName = ThisWorkbook.Sheets("Employees").Cells(2, "C")
```

Figure 41

Notice that the code used the **cells** function to identify the cell. The cells function has two arguments, the first is the row number and the second is the column number or column letter. As shown below, if you use the column letter, you will need to enclose it in double quotes.

Sample code	Description
Cells(Row,Column)	Syntax
Cells(2,1)	Gets data from cell A2 (second row, first column).
Cells(2, "A")	Gets data from cell A2 (second row, column A).

You can also use variables to represent the various arguments. Thus, the following procedure allows the user to pass in a record number and have the code populate the controls for the desired employee.

```
Private Sub LoadEmployeeData(RecordNumber As Long)
    'Create a variable for the row number.
    Dim lngRowNum As Long

    'Use the Excel Match function to calculate the row number of the Employee Record
Number
    lngRowNum = WorksheetFunction.Match(RecordNumber, Range("$A:$A"), 0)

    'Populate the controls
    Me.txtEmployeeNumber = ThisWorkbook.Sheets("Employees").Cells(lngRowNum, "A")
    Me.txtFirstName = ThisWorkbook.Sheets("Employees").Cells(lngRowNum, "B")
    Me.txtLastName = ThisWorkbook.Sheets("Employees").Cells(lngRowNum, "C")
End Sub
```

Code_block 10

The VBA With statement

Notice that the three statements use the same object which refers to the Employees sheet: `ThisWorkbook.Sheets("Employees")`. When the same object is referenced multiple times in a procedure we can use a "**With**" statement to write the object name once instead of multiple times. Thus, instead of referring to our object in each of the three statements, our code can be simplified as follows:

```
Private Sub LoadEmployeeData(RecordNumber As Long)
    'Create a variable for the row number.
    Dim lngRowNum As Long

    'Use the Excel Match function to calculate the row number of the Employee Record
Number.
   lngRowNum = WorksheetFunction.Match(Me.txtEmployeeNumber, Range("$A:$A"), 0)

    'Populate the controls
    With ThisWorkbook.Sheets("Employees")
        Me.txtEmployeeNumber = .Cells(lngRowNum, "A")
        Me.txtFirstName = .Cells(lngRowNum, "B")
        Me.txtLastName = .Cells(lngRowNum, "C")
    End With
End Sub
```

Code_block 11

Later we will add similar statements to populate the other text boxes, the department combo box, and the Is Temp check box.

Because the Region is displayed with multiple option buttons, we will first create a variable to track the value in the region column G, and then we will either create a series of `If… ElseIf` statements or a group of `Select Case` statements as shown below.

```
Dim strRegion As String

strRegion = .Cells(lngRowNum, "G")

If strRegion = "Canada" Then
    Me.optCanada = True
ElseIf strRegion = "Mexico" Then
    Me.optMexico = True
ElseIf strRegion = "Northeast"
    Me.optNortheast = True
ElseIf strRegion = "Northwest"
    Me.optNorthwest = True
ElseIf strRegion = "Southeast"
    Me.optSoutheast = True
ElseIf strRegion = "Southwest"
    Me.optSouthwest = True
End If
```

Code_block 12

```
Dim strRegion As String

strRegion = .Cells(lngRowNum, "G")

Select Case strRegion
    Case "Canada"
        Me.optCanada = True
    Case "Mexico"
        Me.optMexico = True
    Case "Northeast"
        Me.optNortheast = True
    Case "Northwest"
        Me.optNorthwest = True
    Case "Southeast"
        Me.optSoutheast = True
    Case "Southwest"
        Me.optSouthwest = True
End Select
```

Code_block 13

B. Exercise: Populate the Employees form with Excel data

In this exercise you will…

- Add and complete the LoadEmployeeData sub procedure from the previous topic.
- Add multiple navigation buttons so the user can navigate between the records in the spreadsheet.
- Write code to automatically populate the form with the first user's data when the form is initially opened.

1. If needed Start Excel and open your **Employees – WIP.xlsm** workbook.

2. Switch to the Visual Basic Editor and display the code for **frmEmployees**.

3. At the top of the module, just below the **Option Explicit** statement, add the following sub-procedure.

```
Private Sub LoadEmployeeData(RecordNumber As Long)
    'Create various variables.
    Dim lngRowNum As Long          'Long integer for Row Number
    Dim strRegion As String        'String variable for Region

    'Use the Excel Match function to calculate the row number of the
Employee Record Number.
   lngRowNum = WorksheetFunction.Match(Me.RecordNumber, Range("$A:$A"),
0)


    'Populate the controls
    With ThisWorkbook.Sheets("Employees")
        'Populate various controls or variables.
        Me.txtEmployeeNumber = .Cells(lngRowNum, "A")
        Me.txtFirstName = .Cells(lngRowNum, "B")
        Me.txtLastName = .Cells(lngRowNum, "C")
        Me.cboDept = .Cells(lngRowNum, "D")
        Me.txtDependents = .Cells(lngRowNum, "E")
        Me.txtHireDate = .Cells(lngRowNum, "F")
        strRegion = .Cells(lngRowNum, "G")
        Me.chkIsTemp = (.Cells(lngRowNum, "H") = "Yes")
    End With

    'For the region, set the appropriate option button to true.
    Select Case strRegion
        Case "Canada"
            Me.optCanada = True
        Case "Mexico"
            Me.optMexico = True
        Case "Northeast"
            Me.optNortheast = True
        Case "Northwest"
            Me.optNorthwest = True
        Case "Southeast"
            Me.optSoutheast = True
        Case "Southwest"
            Me.optSouthwest = True
    End Select
End Sub
```

4. In the code window, from the Object drop-down menu, choose **UserForm**.

5. In the code window, from the Property drop-down menu, choose **Initialize**.

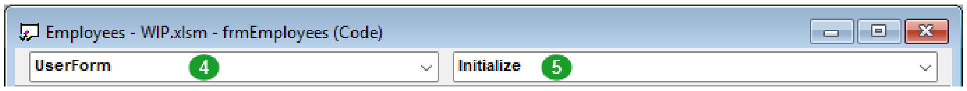


Figure 42

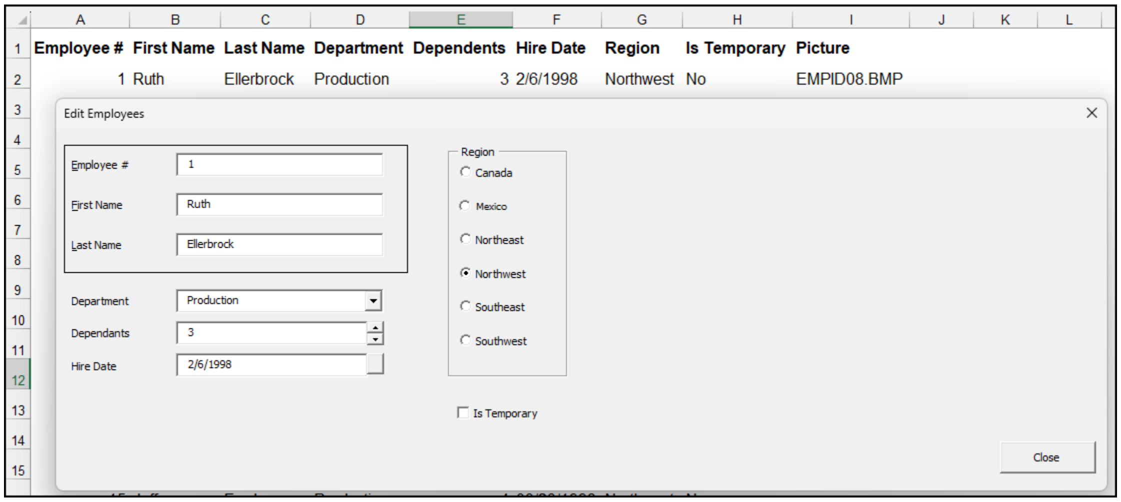
6. Edit the code to call the LoadEmployeeData procedure for the first employee as shown below.

```
Private Sub UserForm_Initialize()
    LoadEmployeeData 1
End Sub
```

Code_block 14

7. Return to Excel and run the ShowForm() macro.

   The record for the first employee, **Ruth Ellerbrock** should be displayed.



Employee #	First Name	Last Name	Department	Dependents	Hire Date	Region	Is Temporary	Picture
1	Ruth	Ellerbrock	Production	3	2/6/1998	Northwest	No	EMPID08.BMP

Edit Employees

Employee #: 1
First Name: Ruth
Last Name: Ellerbrock
Department: Production
Dependants: 3
Hire Date: 2/6/1998

Region: Canada / Mexico / Northeast / [✓ Northwest] / Southeast / Southwest

Is Temporary

Code_block 15

8. Click the **Close** button.

You will now add multiple navigation buttons that let users navigate between the records in the spreadsheet.

9. Add four navigation buttons below the hire date controls and set their **(Names)**, **AutoSize**, and **Captions** as shown in the image and table below.

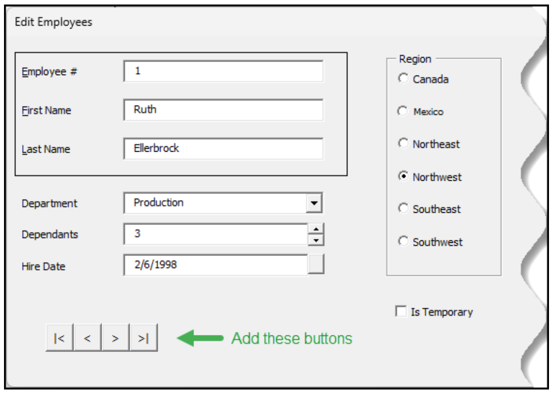


Figure 43

(Name)	Caption	AutoSize
cmdGoToFirst	|<	*True*
cmdGoToPrevious	<	*True*
cmdGoToNext	>	*True*
cmdGoToLast	>|	*True*

Note: The **Captions** for the first and last button are made with the pipe symbol | which is found near the letter P on most keyboards.

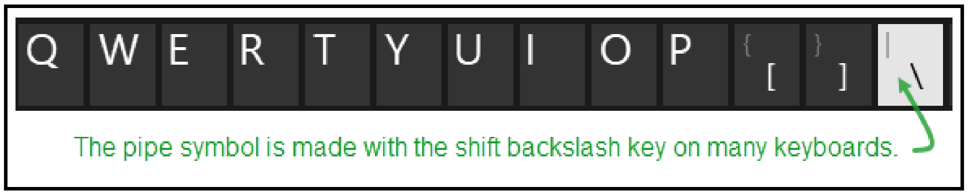

Q W E R T Y U I O P { [}] | \
The pipe symbol is made with the shift backslash key on many keyboards.

Figure 44

10. Go to the code window and enter the following code above the LoadEmployeeData procedure. This will create two function procedures that will be used by our command buttons.

```
Public Function RowCount() As Long
    'Counts number of rows in the current region of Employees!A1.
    RowCount = ThisWorkbook.Sheets("Employees").Range("A1").CurrentRegion.Rows.Count
End Function

Public Function RecordCount() As Long
    'Counts number of records (same as number of rows less 1 for the header)
    RecordCount = RowCount - 1
End Function
```

Code_block 16

11. Go to the code window and enter the following code below the **cmdGetDate_Click()** procedure.

```
Private Sub cmdGoToFirst_Click()
    'Load the first record
    LoadEmployeeData Range("A2")
End Sub
```

```
Private Sub cmdGoToLast_Click()
    Dim lngLastRecord As Long

    'Calculate last record number
    lngLastRecord = WorksheetFunction.Index(Range("$A:$A"), RecordCount
+ 1, 1)

    'Load the last record
    LoadEmployeeData lngLastRecord
End Sub
```

```
Private Sub cmdGoToNext_Click()
    Dim lngRowNum As Long
    Dim lngNextRecord As Long

    'Get row number of current employee.
    lngRowNum = WorksheetFunction.Match(CLng(Me.txtEmployeeNumber),
Range("$A:$A"), 0)

    'If current record is not on the last row, then its okay to go to
the next record.
    If lngRowNum <= RecordCount Then
        lngNextRecord = WorksheetFunction.Index(Range("$A:$A"),
lngRowNum + 1, 1)
        LoadEmployeeData lngNextRecord
    End If
End Sub
```

```
Private Sub cmdGoToPrevious_Click()
    Dim lngRowNum
    Dim lngPrevRecord As Long

    'Get row number of current employee.
    lngRowNum = WorksheetFunction.Match(CLng(Me.txtEmployeeNumber),
Range("$A:$A"), 0)
```

```
    'If the row number is 3 or more, then its is okay to go to previous
record.
    'Why 3?  Because for row 1 and 2 there is no previous employee
record.
    If lngRowNum >= 3 Then
        lngPrevRecord = WorksheetFunction.Index(Range("$A:$A"),
lngRowNum - 1, 1)
        LoadEmployeeData lngPrevRecord
    End If

End Sub
```

Code_block 17

12. Return to Excel and run the ShowForm() macro. Then test the four navigation buttons to verify they work as desired.

13. Close the form and save your workbook.

C. How can I write code to save UserForm data to Excel?

Just as you saw in Topic A that you can use either the **ControlSource** property or VBA code to populate the UserForm controls you can also use these methods to save data from the UserForm to Excel.

If you use the **ControlSource** property method, then the specified Excel cell will automatically change each time the user enters a value into the control. However, this only works if the control is supposed to always populate the same cell.

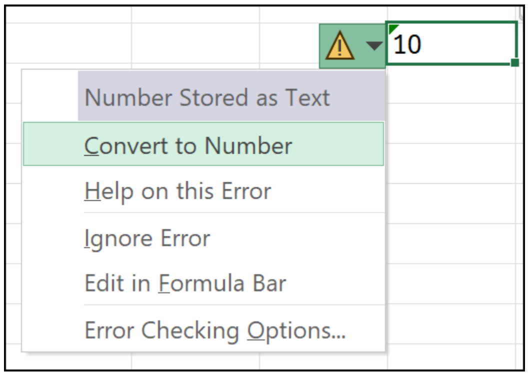
With the VBA code method, you will write code that populates cells based on the values of the various controls.

For example, the code below will save the employee number and names to columns A through C for whichever row is calculated by the Match function.

```
Dim lngRowNum As Long
lngRowNum = WorksheetFunction.Match(Me.txtEmployeeNumber, Range("$A:$A"), 0)


With ThisWorkbook.Sheets("Employees")
    .Cells(lngRowNum, "A") = Me.txtEmployeeNumber
    .Cells(lngRowNum, "B") = Me.txtFirstName
    .Cells(lngRowNum, "C") = Me.txtLastName
End With
```

Code_block 18

Unfortunately, when the Employee Number is entered into column A, Excel will treat the value as a number stored as text because it assumes the values in a text box are text.



Code_block 19

To solve this problem, we tell the code to multiply the text box's value by 1. This forces Excel to recognize that we are dealing with a number, and therefore the cell gets a numeric value. Here is the modified code:

```
Dim lngRowNum As Long
lngRowNum = WorksheetFunction.Match(Me.RecordNumber, Range("$A:$A"), 0)

With ThisWorkbook.Sheets("Employees")
    .Cells(lngRowNum, "A") = Me.txtEmployeeNumber * 1 'Multiply by 1 to force Excel to
recognize it is a number.
    .Cells(lngRowNum, "B") = Me.txtFirstName
    .Cells(lngRowNum, "C") = Me.txtLastName
End With
```

Code_block 20

We can further enhance the code to specify how to format the data. For example the code below shows how you can set the horizontal alignment and the number format for the Employee #.

```
Dim lngRowNum As Long
lngRowNum = WorksheetFunction.Match(Me. RecordNumber, Range("$A:$A"), 0)

With ThisWorkbook.Sheets("Employees")
    .Cells(lngRowNum, "A") = Me.txtEmployeeNumber * 1 'Multiply by 1 to force Excel to
recognize it is a number.
    .HorizontalAlignment = xlRight
    .NumberFormat = "General"
    .Cells(lngRowNum, "B") = Me.txtFirstName
    .Cells(lngRowNum, "C") = Me.txtLastName
End With
```

Code_block 21

To populate the Region from the option buttons you can use a group of If..ElseIf statements like the following:

```
    If Me.optCanada Then
        Cells(lngRowNum, "G") = "Canada"
    ElseIf Me.optMexico Then
        Cells(lngRowNum, "G") = "Mexico"
    ElseIf Me.optNortheast Then
        Cells(lngRowNum, "G") = "Northeast"
    ElseIf Me.optNorthwest Then
        Cells(lngRowNum, "G") = "Northwest"
    ElseIf Me.optSoutheast Then
        Cells(lngRowNum, "G") = "Southeast"
    ElseIf Me.optSouthwest Then
        Cells(lngRowNum, "G") = "Southwest"
    End If
```

Code_block 22

Similarly, to populate the Is Temporary value data in column H with a Yes or No you can use another If statement.

```
    If Me.chkIsTemp Then
        Cells(lngRowNum, "H") = "Yes"
    Else
        Cells(lngRowNum, "H") = "No"
    End If
```

Code_block 23

D. Exercise: Add and save data from the Employees form

In this exercise, you will add a Save button and a New button. You will then add code for both of these buttons.

1. If needed Start Excel and open your **Employees – WIP.xlsm** workbook.
2. Switch to the Visual Basic Editor and display the **frmEmployees**.
3. Add a command button named ***`cmdSave`*** and set its **`Caption`** to ***`Save`*** and set its **`AutoSize`** property to ***`True`***.
4. Double click the **Save** button to open the code window. Then enter the following code.

```
Private Sub cmdSave_Click()
    Dim lngRowNum As Long

    If Me.txtEmployeeNumber <= RecordCount Then
        lngRowNum = WorksheetFunction.Match(CLng(Me.txtEmployeeNumber),
Range("$A:$A"), 0)
    Else
        lngRowNum = RecordCount + 2
    End If

    With Cells(lngRowNum, "A")
        .Value = Me.txtEmployeeNumber * 1 'Multiply by 1 to force Excel to
recognize it is a number.
        .HorizontalAlignment = xlRight
        .NumberFormat = "General"
    End With

    Cells(lngRowNum, "B") = Me.txtFirstName
    Cells(lngRowNum, "C") = Me.txtLastName
    Cells(lngRowNum, "D") = Me.cboDept

    With Cells(lngRowNum, "E")
        .Value = Me.txtDependents * 1
        .HorizontalAlignment = xlRight
        .NumberFormat = "General"
    End With

    With Cells(lngRowNum, "F")
        .Value = Me.txtHireDate
        .HorizontalAlignment = xlLeft
        .NumberFormat = "mm/dd/yyyy"
    End With

    If Me.optCanada Then
        Cells(lngRowNum, "G") = "Canada"
    ElseIf Me.optMexico Then
        Cells(lngRowNum, "G") = "Mexico"
    ElseIf Me.optNortheast Then
        Cells(lngRowNum, "G") = "Northeast"
    ElseIf Me.optNorthwest Then
        Cells(lngRowNum, "G") = "Northwest"
    ElseIf Me.optSoutheast Then
        Cells(lngRowNum, "G") = "Southeast"
    ElseIf Me.optSouthwest Then
        Cells(lngRowNum, "G") = "Southwest"
    End If

    If Me.chkIsTemp Then
        Cells(lngRowNum, "H") = "Yes"
    Else
        Cells(lngRowNum, "H") = "No"
    End If
End Sub
```

Code_block 24

5. Return to the form.

6. Add a command button named ***cmdNew*** and set its **Caption** to ***New*** and set its **AutoSize** property to ***True***.

7. Double click the **New** button to open the code window. Then enter the following code.

```
Private Sub cmdNew_Click()
    'Set the employee number to the current number of records.
    Me.txtEmployeeNumber.Text = RowCount

    'Set the other controls to blank or 0.
    Me.txtFirstName = vbNullString
    Me.txtLastName = vbNullString
    Me.cboDept = vbNullString
    Me.txtDependents = 0

    'Set the hire date to today's date.
    Me.txtHireDate = Date

    'Deselect all the region options.
    Me.optCanada = False
    Me.optMexico = False
    Me.optNortheast = False
    Me.optNorthwest = False
    Me.optSoutheast = False
    Me.optSouthwest = False

    'Uncheck the "Is Temp" check box.
    Me.chkIsTemp = False

    'Move focus to first name.
    Me.txtFirstName.SetFocus
End Sub
```

Code_block 25

8. Return to Excel and run the ShowForm() macro.
Click the **New** button, then fill in the blank boxes as desired.
Then click **Save**. The new employee should be added to the spreadsheet.

9. Close the form and save your workbook.

9. Pictures and Images

Complete this unit and you'll be able to:

A. Describe the features, uses and limitations of an image control

B. List the other controls that can contain a picture.

C. Exercise: Enhance your UserForm with images and pictures

A. What is an image control?

The image control allows you to place an image on your UserForm. This can be used to show decorative elements, logos, icons and other illustrations that can make the UserForm more user friendly and professional.

The image control has a **`Picture`** property. When you select this property, a builder button will appear on the right. Clicking the builder button will let you point to an image file that contains the desired photo or drawing.

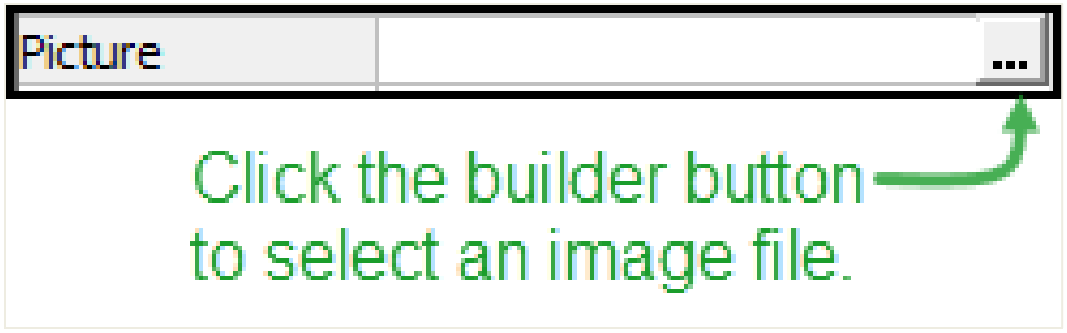


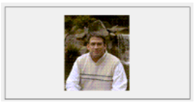
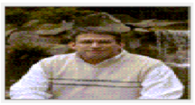
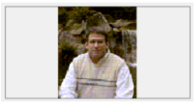
Figure 45 Picture property

The image control supports the following types of image files.

Image type	Extension(s)
Bitmaps	*.bmp, *.dib
GIF Images	*.gif
JPEG Images	*.jpg
Metafiles	*.wmf, *.emf
Icons	*.ico, *.cur

Note:
The image control does not support PNG, TIFF, and SVG images.
These popular image formats cannot be displayed with an image control.

The image control has a **`PictureSizeMode`** property that supports the following values that control how the image is displayed:

Value	Description	Sample
0 – fmPicture SizeModeClip	Shows image at its original size. If you resize the control, the picture inside it will not be resized.	
1 – fmPicture SizeModeStretch	Stretches image to fit control. If you resize the image, the picture inside it will stretch and may become skewed.	
2 – fmPicture SizeModeZoom	Scales image proportionally to fit. If you resize the image, the picture will change size but relative to its original dimensions.	

When the image control is 1) wider and/or taller than the image, and 2) has a either the *0 – fmPictureSizeModeClip* or the *2 – fmPictureSize-ModeZoom* value, the following **PictureAlignment** values can be used to specify where the image should appear within the image control.

0 - frmPictureAlignmentTopLeft		1 - frmPictureAlignmentTopRight
	2 - frmPictureAlignmentCenter	
3 – frmPictureAlignmentBottomLeft		4 - frmPictureAlignmentBottomRight

Like other controls, the image control has an **AutoSize** property that lets the computer automatically resize the image. It also has a **BorderStyle** property and a **BackColor** property that can enhance the look of your image.

B. Do other controls have a picture property?

Yes, many objects have a picture property.   in addition to the image control, the objects listed below have a `Picture` property that can be used to display an image. They may also have additional picture properties as shown in the second column.

Object Type	Other picture related properties

Form	`PictureAlignment`
	`PictureSizeMode`
	`PictureTiling`

Label	`PicturePosition`
ToggleButton	`PicturePosition`

Frame	`PictureAlignment`
	`PictureSizeMode`
	`PictureTiling`

CommandButton	`PicturePosition`

Pages within a MultiPage	`PictureAlignment`
	`PictureSizeMode`
	`PictureTiling`

Image	`PictureAlignment`
	`PictureSizeMode`
	`PictureTiling`

`PictureTiling` will cause a picture to be repeated within its contrainer.

`PicturePosition` is similar to the `PictureAlignment` property but with a few more value options.

C. Exercise: Enhance your UserForm with images and pictures

In this exercise you will do the following:

- Add an image control that will show the selected employee's picture
- Add a text box that will display the file name of the employee's picture
- Add a "Update Picture" command button that will let the user pick a file name that contains the employee's picture
- Add box around the three picture related controls
- Add a calendar icon to the cmdGetDate button
- Write code to populate the image and the file name text box from the Excel worksheet
- Write the code that lets the user update the employee's picture

1. If needed Start Excel and open your **Employees – WIP.xlsm** workbook.

2. Switch to the Visual Basic Editor and display the code for **frmEmployees**.

3. To the right of the Region frame, add an image control named ***imgPicture***. Set the **Height** to *144* pixels and the **Width** to *156* pixels.

4. Below the image control add a text box named ***txtPictureFileName***.

5. To the right of the text box add a command button named ***cmdUpdatePicture*** and set its **Caption** to ***Update Picture.***

6. Draw a label around the three other picture related controls. Set its **(Name)** to box, erase its **Caption**, set its **BackStyle** to ***0 – frmBackStyleTransparent*** and its **BorderStyle** to ***1 – frmBorderStyleSingle***.

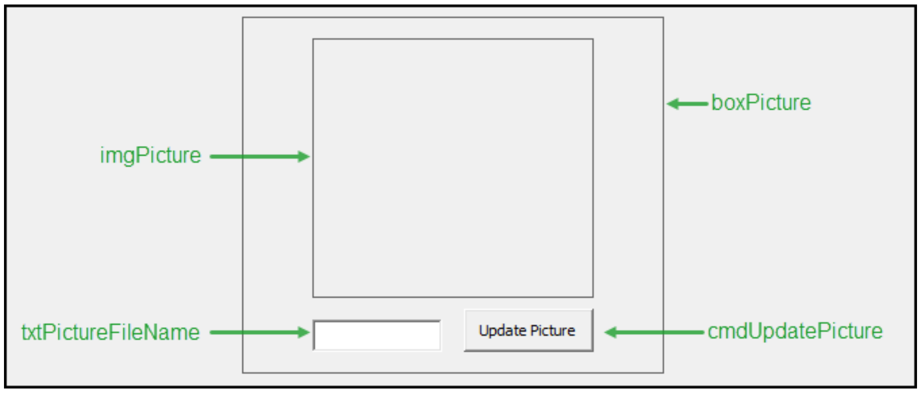


Figure 46

7. Place the ***boxPicture*** behind the other controls by clicking on it and then choosing **Format** > **Order** > **Send to Back** from the menu.

8. Select the ***cmdGetDate*** button and click on its **Picture** property. Then click the builder button on the right side.

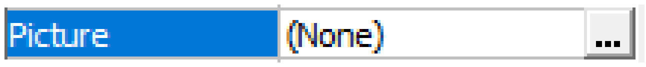


Figure 47

When prompted, choose the **One.bmp** image file from the class files folder and click **Open**. Finally resize the button as desired.

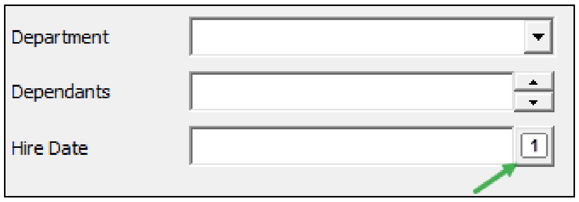


Figure 48

You will now write the code to populate the image and the file name text box from the Excel worksheet

9. Open Module1 and at the top of the module, below the Option Explicit statement, add the following function which will return the folder, but not the file name, of the active workbook.

```
Public Function ActiveWorkbookFolder() As String
    'This function calculates the path, but not the filename of the active
workbook.

    'Create variables.
    Dim strPathAndFileName As String 'String variable that tracks the path and
file name.
    Dim i As Integer                 'Integer variable used for counting.

    'Get the full path and filename of the ActiveWorkbook.
    strPathAndFileName = ThisWorkbook.FullName

    'Starting with the last character and looping toward the first of the
strPathAndFileName...
    For i = Len(strPathAndFileName) To 1 Step -1
        'If the selected character is a backslash, then...
        If Mid(strPathAndFileName, i, 1) = "\" Then
            '... the folder is everthing up to and including the backslash.
            ActiveWorkbookFolder = Left(strPathAndFileName, i)
            'Exit the loop
            Exit For
        End If
    Next i
End Function
```
Code_block 26

10. Now switch to the frmEmployees' code window and locate the LoadEmployeeData sub-procedure.

11. Add a third dimension statement for a new variable named **strPicPath**.

```
Private Sub LoadEmployeeData(RecordNumber As Long)
    'Create various variables.
    Dim lngRowNum As Long         'Long integer for Row Number
    Dim strRegion As String       'String variable for Region
    Dim strPicPath As String      'String variable for Picture
```
Code_block 27

12. In the section where you populate the controls, add code to populate the strPicPath from the worksheet, and then populate the txtPictureFilename from the variable.

```
    'Populate the controls
    With ThisWorkbook.Sheets("Employees")
        'Populate various controls or variables.
        Me.txtEmployeeNumber = .Cells(lngRowNum, "A")
        Me.txtFirstName = .Cells(lngRowNum, "B")
        Me.txtLastName = .Cells(lngRowNum, "C")
        Me.cboDept = .Cells(lngRowNum, "D")
        Me.txtDependents = .Cells(lngRowNum, "E")
        Me.txtHireDate = .Cells(lngRowNum, "F")
        strRegion = .Cells(lngRowNum, "G")
        Me.chkIsTemp = (.Cells(lngRowNum, "H") = "Yes")
        strPicPath = .Cells(lngRowNum, "I")
        Me.txtPictureFileName = strPicPath
    End With
```

Code_block 28

13. Next add the following code to conditionally load the picture into the image.

```
    'If the variable strPicPath has a value (has a length) then ....
    If Len(strPicPath) Then

        'Use Excel's LoadPicture method to populate the image's picture.
        Me.imgPicture.Picture = LoadPicture(ActiveWorkbookFolder & strPicPath)

        'Make the image visible.
        Me.imgPicture.Visible = True

    Else    'If strPicPath has no value then erase the picture by using the
LoadPicture method with a blank (null) value.
        Me.imgPicture.Picture = LoadPicture(vbNullString)
    End If
```

Code_block 29

You will now write code so the user can click the Update Picture button and then choose an image for the selected employee.

14. Return to the frmEmployees form and double click the Update Picture button.

15. Add the following code which will prompt the user to pick a file for the selected employee's picture.

```
Private Sub cmdUpdatePicture_Click()
    'Create a variable to track the file name.
    Dim strFileName As String

    'If an error occurs, go to the ErrHandler label.
    On Error GoTo ErrHandler

    'Change the directory to the ActiveWorkbookFolder
    ChDir ActiveWorkbookFolder
    'Using Excel's GetOpenFilename method, prompt the user for an image file.
    strFileName = Application.GetOpenFilename("Image files
(*.bmp;*.gif;*.jpg;*.jpeg;*.wmf;*.emf;*.ico;*.dib;*.cur),*.bmp;*.gif;*.jpg;*.jpe
g;*.wmf;*.emf;*.ico;*.dib;*.cur")

    'If an image file was chosen then ...
    If Len(strFileName) Then
        Me.imgPicture.Picture = LoadPicture(strFileName)     'Load the picture
        Me.imgPicture.Visible = True                         'Make the image
control visible
        Me.txtPictureFileName = FileNameOnly(strFileName)    'Put the filename in
the Picture Filename text box.
    End If

    Exit Sub

ErrHandler:
    Select Case Err.Number
        Case 481
            MsgBox strFileName _
                & vbNewLine & " is not a valid picture file.", vbCritical
        Case Else
            MsgBox Err.Description, vbCritical, Err.Number
    End Select
End Sub
```

Code_block 30

Note that the above code uses the **FileNameOnly()** function to populate the txtPictureFileName. This is not a built-in function. Therefore, you will now create this function in Module1.

16. Open Module1 and add the FileNameOnly function.

```
Public Function FileNameOnly(FullPathAndFileName As String)
    Dim i As Integer    'Create a counter variable

    'Loop from the last character of the FullPathAndFileName  to the first.
    For i = Len(FullPathAndFileName) To 1 Step -1
        'If the current character is a backslash then ...
        If Mid(FullPathAndFileName, i, 1) = "\" Then
            '... File name is the text after the backslash
            FileNameOnly = Mid(FullPathAndFileName, i + 1)
            'Exit the loop
            Exit For
        End If
    Next i
End Function
```

Code_block 31

You will now update the cmdNew_Click() to clear out the image when a new record is created.

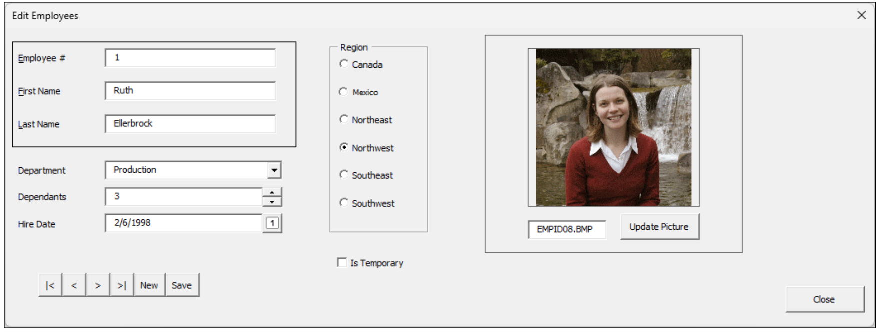
17. Return to the code for frmEmployees and add the following code after me.txtDependents = 0.

```
    Me.txtDependents = 0
    Me.txtPictureFileName = vbNullString
    Me.imgPicture.Picture = LoadPicture(vbNullString)
```

Code_block 32

18. Return to Excel and run the ShowForm macro.

    The image should be loaded, and the text box below the image should show the name of the image file.



Edit Employees

Employee # 1
First Name Ruth
Last Name Ellerbrock
Department Production
Dependants 3
Hire Date 2/6/1998

Region
Canada [ ]
Mexico [ ]
Northeast [ ]
Northwest [✓]
Southeast [ ]
Southwest [ ]

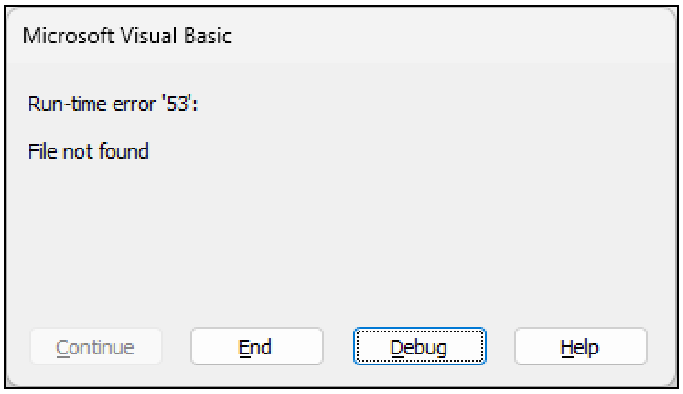
Is Temporary [ ]

EMPID08.BMP

Figure 49

19. Click the various navigation buttons to verify the other employee pictures load correctly. Then click the **Close** button to unload the form.

Unfortunately, our code does have a bug. If the picture file listed in column I does not exist in our folder, the following error will occur.



Microsoft Visual Basic

Run-time error '53':

File not found

Figure 50

To prevent this error you will now add some error handling code, so the user doesn't get the following error message.

20. Return to the LoadEmployeeData sub-procedure.

21. Below the **Dim** statements, add the following statement which tells Excel to go to the ErrHandler label.

```
     Dim strRegion As String      'String variable for Region
     Dim lngRowNum As Long        'Long integer for Row Number
     Dim strPicPath As String     'String variable for Picture

     'If an unexpected error occurs, code continue at the ErrHandler.
     On Error GoTo ErrHandler
```

Code_block 33

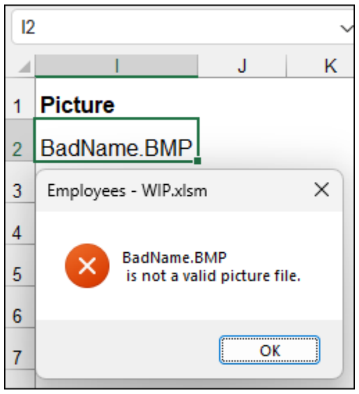
22. Now add the following Error Handling code before the End Sub. As shown in the comments, this code will let the user know that a file name is invalid if the file is not found and/or if the path is not found.

```
    'If the code gets this far, everything worked. Therefore, exit the sub-
procedure so the Error Handler doesn't run.
    Exit Sub


ErrHandler: 'This is a label.  Earlier coded instructed to code to go here if
an error occurred.

    'Process errors based on their error number.
    Select Case Err.Number

        Case 53, 76 'Error 53 is File Not Found, and 76 is Path Not Found.
            'If a filename was provided, tell the user it is not valid.
            If Me.txtPictureFileName <> vbNullString Then
                MsgBox txtPictureFileName & vbNewLine _
                    & " is not a valid picture file.", _
                    vbCritical, ThisWorkbook.Name
            End If
            Resume Next 'Return control to the next line of code.

        'If any other error occurs, let the user know.
        Case Else
            MsgBox "Error #" & Err.Number _
                  & Err.Description, _
                  vbCritical, ThisWorkbook.Name
            Resume Next 'Return control to the next line of code.
    End Select
End Sub
```

Code_block 34

23. Return to Excel and purposely change the Picture in cell I2 to BadName.BMP. Then run the ShowForm macro.

You should see the following message.



BadName.BMP
is not a valid picture file.

Figure 51

Click **OK** and you should then see the Employee form.

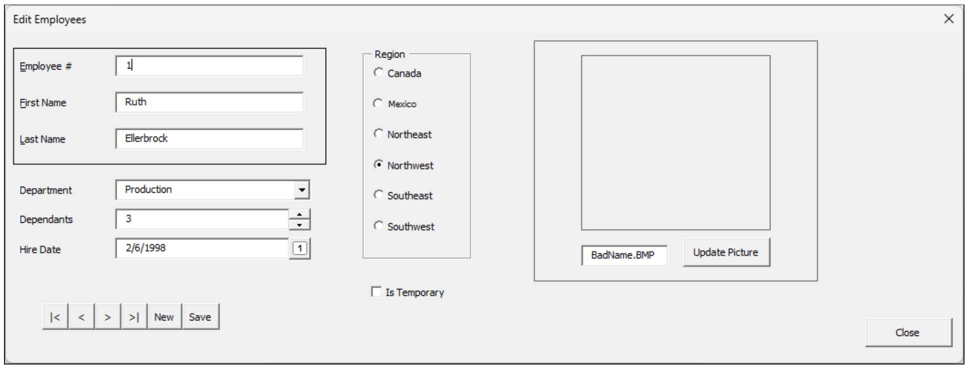


Figure 52

24. Click the various navigation buttons to verify the other employee pictures load correctly. Then click the **Close** button to unload the form.

25. Change cell I2 back to EMPID08.BMP and then save your workbook.

10. The RefEdit controls

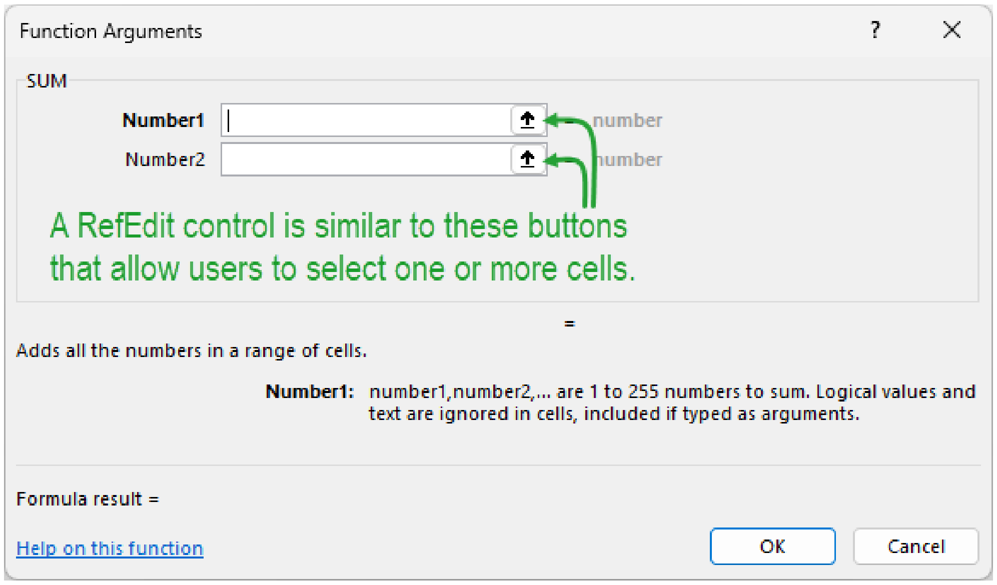
Complete this unit and you'll be able to:

A. Describe the features and uses of RefEdit control

B. Exercise: Enhance your UserForm with a RefEdit control

A. What is a RefEdit control?

A RefEdit control is a control that allows the user to select a range of cells. It is similar to the button used by the Function Arguments dialog box as shown below.


Function Arguments

SUM

Number1 | number

Number2 | number

A RefEdit control is similar to these buttons that allow users to select one or more cells.

=

Adds all the numbers in a range of cells.

Number1: number1,number2,... are 1 to 255 numbers to sum. Logical values and text are ignored in cells, included if typed as arguments.

Formula result =

Help on this function

OK Cancel

Figure 53

As shown below, it is best to provide a label next to the RefEdit control to identify its use.

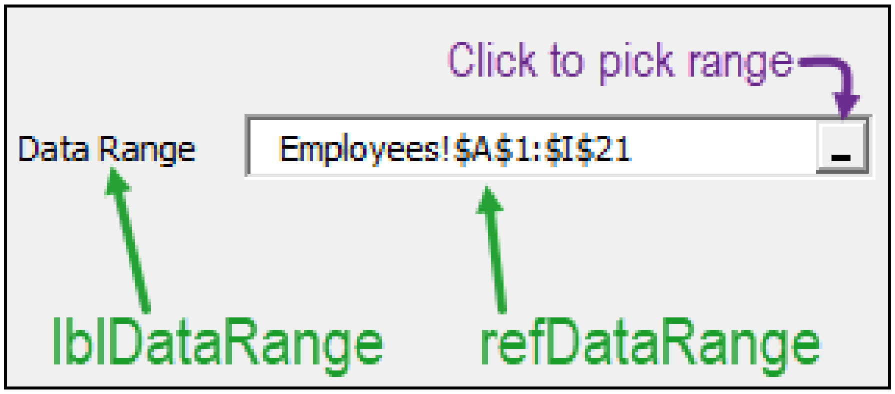


Figure 54

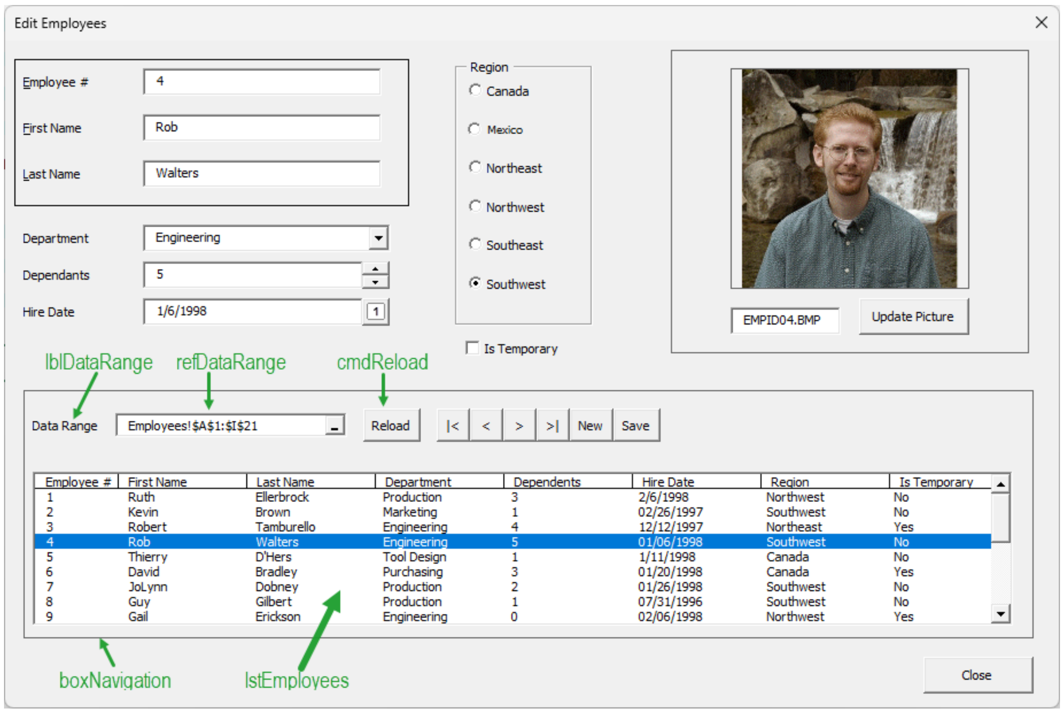
Notes and Limitations:

- The RefEdit control is **NOT** available in the 64-bit versions of Microsoft Office / 365 applications.
- The RefEdit control sometimes yields unpredictable, undesirable results, especially when multiple UserForms and/or workbooks are open.

B. Exercise: Add a RefEdit control to your UserForm

In this exercise, you will do the following to enhance your UserForm with a RefEdit control and a few other controls as shown below.

- Add a RefEdit control to let the user define the range for the list of employees
- Add a list box to show the selected list of employees and their department
- Add a control button to force the list box to update the list of employees
- Create a sub-procedure to use when updating the list of employees
- Write code to update the list of employees when the form first loads
- Write code to synchronize the list box with the other controls



Employee #: 4
First Name: Rob
Last Name: Walters
Department: Engineering
Dependants: 5
Hire Date: 1/6/1998
Region: Southwest
Data Range: Employees!$A$1:$I$21

Employee #	First Name	Last Name	Department	Dependents	Hire Date	Region	Is Temporary
1	Ruth	Ellerbrock	Production	3	2/6/1998	Northwest	No
2	Kevin	Brown	Marketing	1	02/26/1997	Southwest	No
3	Robert	Tamburello	Engineering	4	12/12/1997	Northeast	Yes
4	Rob	Walters	Engineering	5	01/06/1998	Southwest	No
5	Thierry	D'Hers	Tool Design	1	1/11/1998	Canada	No
6	David	Bradley	Purchasing	3	01/20/1998	Canada	Yes
7	JoLynn	Dobney	Production	2	01/26/1998	Southwest	No
8	Guy	Gilbert	Production	1	07/31/1996	Southwest	No
9	Gail	Erickson	Engineering	0	02/06/1998	Northwest	Yes

Figure 55

1. Add a label named ***lblDataRange*** and set its **Caption** to ***Data Range***.

2. Add a RefEdit control and name it ***refDataRange***.

3. Add a command button named ***cmdReload*** and set its **Caption** to ***Reload***.

4. Add a list box named ***lstEmployees*** and set its **ColumnCount** to ***8***. Do NOT set its **RowSource** property as that will be set with code.

5. Move the navigation buttons next to the Reload button.

6. Add a label and set its name to ***boxNavigation***. Set the **BackStyle** property to ***1 – frmBackStyleTransparent*** and the **BorderStyle** to ***1 – frmBorderStyleSingle***.

7. Go to the code for the Employee form and add the following sub-procedure above the cmdClose_Click event.

```
Private Sub UpdateEmployeeList()
    Dim strWorksheetName As String          'Variable for the name of the
worksheet
    Dim wks As Worksheet                    'Object variable to represent the
worksheet
    Dim strRangeAddress As String           'Variable for the address of the
range of data cells.
    Dim strDataRange As String              'Variable for the complete address
(worksheet plus range addresses) of the data range
    Dim exclMarkPos As Long                 'Numeric variable for the location
of the exclamation mark character in refDataRange
    Dim rng As Range                        'Object variable for the data
range.

    ' Check if refDataRange is empty then...
    If Trim(Me.refDataRange) = "" Then
        ' Use first worksheet and its CurrentRegion from A1
        strWorksheetName = ThisWorkbook.Worksheets(1).Name          'Get the
name of the first worksheet
        Set wks = ThisWorkbook.Worksheets(strWorksheetName)         'Creates an
object variable that represents the worksheet.
        strRangeAddress = wks.Range("A1").CurrentRegion.Address     'Get the
address of the range of data cells starting with A1.
        strDataRange = strWorksheetName & "!" & strRangeAddress     'Data range
is the worksheet name plus an exlcamation mark plus the range name.
        Me.refDataRange = strDataRange                              'Set the
RefEdit control to the strDataRange.

    Else
        'If the RedEdit control is populated then...
        exclMarkPos = InStr(Me.refDataRange, "!")                   'Determine
the location of the exclamation mark

        'If exclamation mark is there, then ....
        If exclMarkPos > 0 Then
            strWorksheetName = Left(Me.refDataRange, exclMarkPos - 1)
'strWorksheetName is text before the exclamation mark
            strRangeAddress = Mid(Me.refDataRange, exclMarkPos + 1)
'strRangeAddress is best after the exclamation mark

        'If exclamation mark is missing so ....
        Else
            strWorksheetName = ThisWorkbook.Worksheets(1).Name      'No
exclamation mark so get name of first worksheet.
            strRangeAddress = Me.refDataRange
'strRangeAddress comes from the RefEdit control
        End If

        strDataRange = Me.refDataRange
'strDataRange is determined from the RefEdit control.
    End If
    ' Set worksheet and range
    Set wks = ThisWorkbook.Worksheets(strWorksheetName)
    Set rng = wks.Range(strRangeAddress)

    'Set the lstEmployees RowSource.
    'Exclude header row via Offset(1,0).
    'Resize the results to include all the rows less one (since the header is
not included)
    'And include exactly 8 columns.
    Me.lstEmployees.RowSource = strWorksheetName & "!" & _
        rng.Offset(1, 0).Resize(rng.Rows.Count - 1, 8).Address
End Sub
```

Code_block 35

8. In the LoadEmployeeData sub-procedure, just above the Exit sub procedure, add the following code to keep the list box in sync with the loaded employee's record number.

```
    'In the employee list box, hightlight the row that matches the current
employee's record number.
    Me.lstEmployees = RecordNumber

    'If the code gets this far, everything worked. Therefore exit the sub-
procedure so the Error Handler doesn't run.
    Exit Sub
```

Code_block 36

9. In the UserForm_Initialize event, add the following statement to update the Employee list box.

```
Private Sub UserForm_Initialize()
    UpdateEmployeeList
    LoadEmployeeData 1
End Sub
```

Code_block 37

10. Now write code to load an employee when the user selects the employee from the list box. To do so, at the top of the screen, from the Object drop-down, select the **lstEmployees** object. Then from the Procedure drop-down, select the **AfterUpdate** event. Finally write the code shown below:

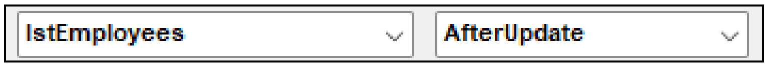


Figure 56

```
Private Sub lstEmployees_AfterUpdate()
    LoadEmployeeData Me.lstEmployees
End Sub
```

Code_block 38

11. Write code to update the employee list box every time the user clicks the Reload button. First select cmdReload from the Object drop-down, and then, if needed, select the Click event from the Procedure drop-down.

```
Private Sub cmdReload_Click()
    UpdateEmployeeList
End Sub
```

Code_block 39

12. Write code in the cmdSave_Click() procedure to temporarily clear out the refDataRange value and then re-update the Employee list box.

```
    Me.refDataRange = vbNullString
    UpdateEmployeeList
```

Code_block 40

13. Return to Excel and run the ShowForm() macro, then test the following:

    Does the RefEdit control show the range of cells that has employee data?

    Does the list box show a list of the employees and their information?

    Do the various text boxes, combo boxes, etc. show information about the selected employee?

    If you click the RefEdit button and select a smaller range like A1:H7, and then click the Reload button, does the list box show you less employees?

    When you click on an employee on the list box, does that employee's information populate the various controls?

    If you click a navigation button do the controls show the next or previous employee's information, and does the list box highlight that employee?

11. Open Lab

With the help of your instructor, create a project of your own choosing or select one of the projects listed below. As you do so, remember to implement the five-step UserForm creation process from Lesson 2, Topic C which is summarized below:

1. **Plan** your form's look, feel and functionality.
2. **Draw** the form and its controls.
3. **Set properties** for each control.
4. **Write VBA code**.
5. **Test, debug and fine-tune** as needed.

Project: Time picker:

Purpose:
Allow uses to easily pick a time of day that will be entered into the active cell.

Features:

- Option buttons for a clock type: 24 hour or 12 hour.
- Combo box or list box for hours (12 or 24 options depending on the clock type.
- Combo box for minutes (0 to 59).
- Label showing the user the time they have picked.
- Command button to populate the active cell and then close the form.
- Command button to cancel the input and close the form.

Project: Employee Onboarding Tracker

Purpose:
Enhance the project created in lessons one through ten to streamline the process of entering and tracking new employee information.

Features:

- Add text boxes for job title, email address, office extension, and personal phone number.

- Add option buttons for employment type (full-time, part-time, intern, contractor).
- Check boxes to track when the following onboarding steps are completed: ID Badge, Orientation, Other paperwork.

Project: Expense Reimbursement Form

Purpose:
Allow employees to submit expense claims with itemized entries.

Features:
- Text boxes for employee name and ID.
- Combo box for employee department.
- List box to display added expense items.
- Text box for expense information including date, amount, vendor, and item purchased.
- Option buttons for payment method (Cash, Credit, Direct Deposit).
- Label that shows total expenses.
- Code to populate Excel spreadsheet.

Project: Inventory Management Tools

Purpose:
Help businesses track stock levels and reorder needs.

Features:
- ComboBox to select product category.
- ListBox to display current inventory items.
- TextBoxes for item name, quantity, reorder level, and supplier.
- CheckBox to mark items as discontinued.
- CommandButtons to add/update/delete inventory items.
- Conditional formatting or color-coded labels to indicate low stock.

Project: Customer Feedback Collector

Purpose:
Gather and store customer feedback for service improvement.

Features:
- TextBoxes for customer name and email.
- ComboBox for service/product used.
- OptionButtons for satisfaction rating (1–5).

- Multi-line TextBox for comments.
- CheckBox to opt-in for follow-up.
- CommandButton to submit feedback and clear the form.
- Label to confirm successful submission.