Author: Hossou Zounffa

VBA Script For Adding A Command Button To A Userform

2/23/2024

Code #1

```
Sub AddCommandButton(frm As Object, ByVal name As String, ByVal caption As String, _
                ByVal Left As Single, ByVal Top As Single, ByVal Width As Single, _
                Optional ByVal Height As Single = 20)
   Dim cmdBtn As Object
   Set cmdBtn = frm.Designer.Controls.Add("Forms.CommandButton.1")
   cmdBtn.name = name
   cmdBtn.caption = caption
   cmdBtn.Left = Left
   cmdBtn.Top = Top
   cmdBtn.Width = Width
   cmdBtn.Height = Height
End Sub
```

Why is Height optional in this code, and what does it mean?

Height is optional with a default value, allowing the subroutine to be called without specifying this parameter. The control will use the default height unless otherwise specified.

Despite the optional parameter, what happens if I specify the height in the call? Does it override no matter what value of height I use, or it overrides relative to 20, which is the optional value?

When you specify the height in the subroutine call, the value you provide will directly set the height of the command button, overriding the default value of 20 completely, regardless of what the specified value is. The override is absolute, not relative to the default value; this means if you specify a height of 30, the button's height will be 30, not 50 (20 + 30).

What if I specify a value under 20, like 10?

If you specify a value under 20, such as 10, for the height in the subroutine call, the command button's height will be set to that specified value, in this case, 10. The override applies regardless of whether the specified value is above or below the default value of 20. This allows for precise control over the appearance of the button according to the specific requirements of your UserForm's design.

Can we set an optional value for all the parameters in the code, if not why?

Not all parameters in a subroutine or function can be made optional in VBA. Making a parameter optional is useful when you want to allow a caller to omit that parameter, providing flexibility. However, for the code to execute correctly without errors, any parameters that follow an optional parameter must also be optional. This is because parameters are positional; you cannot skip an optional parameter to supply a value for a subsequent non-optional parameter. If essential parameters that determine the core functionality of the subroutine could be omitted, it might lead to runtime errors or unintended behavior. Therefore, while you can make many parameters optional, critical ones that ensure the subroutine functions correctly should not be optional.

In code #1, what are the critical parameters that ensure the subroutine functions correctly and should not be optional?

Critical parameters include the form object (**frm**), control names (**name**), and dimensions/positions (**Left**, **Top**, **Width**) as they are fundamental for creating and placing the control.

What is the difference between the code #1 and code 2# below?

The first code snippet uses frm to refer to the UserForm object and directly sets properties within the subroutine call parameters. The second snippet utilizes a With block to set properties. Both approaches achieve the same result but differ slightly in syntax. The choice between them depends on personal or project-specific naming conventions and whether you prefer to set properties inline or within a With block.

Code #2

```
Sub AddCommandButton(frm As Object, ByVal name As String, ByVal caption As String, _
                ByVal Left As Single, ByVal Top As Single, ByVal Width As Single, _
                Optional ByVal Height As Single = 20r)
   Dim btn As Object
      Set cmdBtn = frm.Designer.Controls.Add("Forms.CommandButton.1")
   With cmdBtn
      .name = name
      .caption = caption
      .Left = Left
      .Top = Top
      .Width = Width
   End With
End Sub
```

What does ByVal mean in the first code?

In VBA, ByVal in the function or subroutine declaration specifies that the argument is passed by value. This means that the subroutine or function receives a copy of the argument's value. Modifications to this parameter within the subroutine or function do not affect the original variable's value in the calling code. It is a way to ensure that the original data passed to the subroutine or function remains unchanged, safeguarding against unintended side effects of the subroutine or function on the original data.

If I remove ByVal keyword from the code, what happens?

If you remove the ByVal keyword from the code, the parameters will default to ByRef if not explicitly specified otherwise. ByRef means the argument is passed by reference, allowing the subroutine to modify the original variable's value used in the call. In most cases with simple data types like String, Integer, or Single, and when modifying properties of an object (like setting properties of a control), the practical difference in behavior might not be noticeable. However, it's crucial for understanding how data is passed around, especially with complex data types or when the intention is to modify the input data directly.

I still don't understand the importance of ByVal.

ByVal is important for ensuring that modifications within a subroutine do not unintentionally alter the original variables passed from the calling code, maintaining data integrity.

A copy of the variable's value, which variable in my case?

In your case, it refers to any variable passed to the subroutine with **ByVal**, such as the **name**, **caption**, **Left**, **Top**, or **Width** parameters.

Give a real-world example.

```
Sub PreviewNameChange(ByVal originalName As String, ByVal newName As String)
originalName = newName
MsgBox "Previewing change to: " & originalName
End Sub
```

Provide two examples, one where ByVal is present and prevents changes, and another where ByVal is missing, and changes happen.

Example 1: ByVal Prevents Changes

```
Sub TestByVal()
Dim originalValue As Integer
originalValue = 10
Call ChangeValueByVal(originalValue)
MsgBox "The original value after ByVal call is: " & originalValue
End Sub

Sub ChangeValueByVal(ByVal value As Integer)
value = 20
End Sub
```

Running **TestByVal** will show that **originalValue** remains 10 despite the attempt to change it within **ChangeValueByVal**

Example 2: Missing ByVal (Defaults to ByRef) Allows Changes

```
Sub TestByRef()
Dim originalValue As Integer
originalValue = 10
Call ChangeValueByRef(originalValue)
MsgBox "The original value after ByRef call is: " & originalValue
End Sub

Sub ChangeValueByRef(value As Integer)
value = 20
End Sub
```

In this case, running **TestByRef** will show that **originalValue** changes to 20 because the parameter is passed by reference, allowing **ChangeValueByRef** to modify the original variable's value.

What happens if you execute/run code #1/ #2 by itself?

Executing the AddCommandButton subroutine by itself won't have any visible effect because it's a helper function designed to be called from another piece of code where a UserForm (frm) object is defined and passed as an argument. This subroutine adds a command button to a given UserForm, but without calling this subroutine and providing the necessary arguments (a UserForm object, name, caption, left, top, width, and height of the button), no command button will be added to any UserForm. For the subroutine to work, it needs to be called within the context of an existing or newly created UserForm in VBA.

Provide an analogy to better explain the idea of executing the AddCommandButton subroutine by itself.

Executing the **AddCommandButton** subroutine by itself, without integrating it into a broader context where it's actually called and utilized, is like placing a ruler in the air and expecting it to measure distance without anchoring it to a start and end point. The ruler (subroutine) has the potential to measure and position a figure (add a button), on a physical object, but without application (calling the subroutine code #1/#2 with a specific UserForm and parameters), it remains an unutilized tool, floating without purpose or effect.

Provide an example of how to call the AddCommandButton subroutine for a UserForm.

Assume My UserForm is already created and referenced.

Example call to AddCommandButton

```
AddCommandButton frm, "CommandButton_Submit", "Submit all selections", 10, 230, 180, 30.
```

	CommandButton_Submit: name of the button
	Submit all selections: caption of the button
10	The left position
230	The top position of the button
180	The width
30	The height of the button (optional, defaults to 20 if not provided)

Write a code to create and display a UserForm that includes a command button using the AddCommandButton subroutine.

Example #1

```
Sub AddCmb()
Dim frm As Object
Set frm = ThisWorkbook.VBProject.VBComponents.Add(3) ' 3 denotes a UserForm

frm.Properties("Caption") = "Publications List"
frm.Properties("Width") = 400
frm.Properties("Height") = 400

' Parameters are: Form, Name, Caption, Left, Top, Width, Optional Height
AddCommandButton frm, "CommandButton_Submit", "Submit all selections", 10, 230, 180, 30

VBA.UserForms.Add(frm.name).Show
End Sub
```

Example #2

```
Sub AddCmb()
Dim frm As Object
Set frm = ThisWorkbook.VBProject.VBComponents.Add(3) ' 3 denotes a UserForm

frm.Properties("Caption") = "Publications List"
frm.Properties("Width") = 400
frm.Properties("Height") = 400

' Parameters are: Form, Name, Caption, Left, Top, Width, Optional Height

AddCommandButton frm, "CommandButton_Submit1", "Submit all selections1", 10, 230, 180

AddCommandButton frm, "CommandButton_Submit2", "Submit all selections2", 10, 200, 180

AddCommandButton frm, "CommandButton_Submit3", "Submit all selections3", 10, 170, 180

VBA.UserForms.Add(frm.name).Show
End Sub
```