Every programmer needs a Rubberduck

'Apply' logic for UserForm dialog

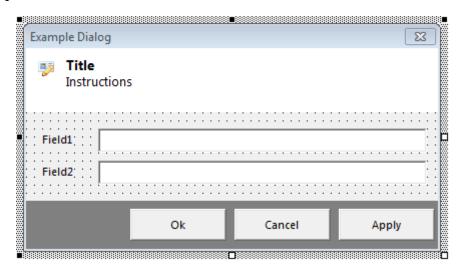
Posted on May 8, 2018May 8, 2018 by Rubberduck VBA

A recent comment on <u>UserForm1.Show</u>

(https://rubberduckvba.wordpress.com/2017/10/25/userform1-show/) asked about how to extend that logic to a dialog that would have an "Apply" button. This article walks you through the process – and this time, there's a download link

(https://www.dropbox.com/s/x9sty06m2xh5zb5/ExampleDialog.xlsm?dl=0)!

The dialog is a simple UserForm with two textboxes and 3 buttons:



The **Model** for this dialog is a simple class exposing properties that the two textboxes manipulate – I've named the class ExampleModel:

```
1
     Option Explicit
 2
 3
     Private Type TModel
 4
         field1 As String
 5
         field2 As String
6
     End Type
7
8
     Private this As TModel
9
10
     Public Property Get field1() As String
         field1 = this.field1
11
12
     End Property
13
14
     Public Property Let field1(ByVal value As String)
15
         this.field1 = value
16
     End Property
17
18
     Public Property Get field2() As String
19
         field2 = this.field2
     End Property
20
21
     Public Property Let field2(ByVal value As String)
22
23
         this.field2 = value
     End Property
24
```

I also defined a simple <code>IDialogView</code> interface, which can be implemented by any other dialog, since it passes the model as an <code>Object</code> (i.e. it's not tightly coupled with the <code>ExampleModel</code> class in any way); the contract is simply "here's your model, now show me a dialog and tell me if I can proceed to consume the model" – in other words, the caller provides an instance of the model, and the implementation returns <code>True</code> unless the user cancelled the form.

```
Option Explicit

Public Function ShowDialog(ByVal viewModel As Object) As Boolean
End Function
```

The form's code-behind therefore needs to implement the <code>IDialogView</code> interface, and somehow store a reference to the <code>ExampleModel</code>. And since we have cancellation logic but we're not exposing it (we don't need to – the <code>IDialogView.ShowDialog</code> interface handles that concern, by returning <code>False</code> if the dialog is cancelled), the <code>IsCancelled</code> flag is just internal state.

As far as the "apply" logic is concerned, the thing to note here is the Public Event ApplyChanges event, which we *raise* when the user clicks the "apply" button:

```
1
     Option Explicit
 2
 3
     Public Event ApplyChanges(ByVal viewModel As ExampleModel)
 4
 5
     Private Type TView
 6
         IsCancelled As Boolean
 7
         Model As ExampleModel
 8
     End Type
 9
     Private this As TView
10
11
     Implements IDialogView
12
13
     Private Sub AcceptButton Click()
14
         Me.Hide
     End Sub
15
16
17
     Private Sub ApplyButton Click()
18
         RaiseEvent ApplyChanges(this.Model)
19
     End Sub
20
21
     Private Sub CancelButton Click()
22
         OnCance1
23
     End Sub
24
     Private Sub Field1Box_Change()
25
         this.Model.field1 = Field1Box.value
26
27
     End Sub
28
29
     Private Sub Field2Box Change()
         this.Model.field2 = Field2Box.value
30
31
     End Sub
32
33
     Private Sub OnCancel()
34
         this.IsCancelled = True
35
         Me.Hide
     End Sub
36
37
38
     Private Function IDialogView_ShowDialog(ByVal viewModel As Object) As Boole
         Set this.Model = viewModel
39
40
         Me.Show vbModal
         IDialogView ShowDialog = Not this.IsCancelled
41
42
     End Function
43
44
     Private Sub UserForm Activate()
45
         Field1Box.value = this.Model.field1
46
         Field2Box.value = this.Model.field2
47
     End Sub
48
49
     Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
50
         If CloseMode = VbQueryClose.vbFormControlMenu Then
51
             Cancel = True
52
             OnCancel
53
         End If
54
     End Sub
```

The **Presenter** class does all the fun stuff. Here I've decided to allow the model's data to be optionally supplied as parameters to the Show method; the form handles its Activate event to make sure the form controls reflect the model's initial values when the form is displayed:

```
1
     Option Explicit
 2
     Private WithEvents view As ExampleDialog
 3
 4
     Private Property Get Dialog() As IDialogView
 5
         Set Dialog = view
6
     End Property
7
8
     Public Sub Show(Optional ByVal field1 As String, Optional ByVal field2 As S
9
10
         Set view = New ExampleDialog
11
12
         Dim viewModel As ExampleModel
         Set viewModel = New ExampleModel
13
14
         viewModel.field1 = field1
         viewModel.field2 = field2
15
16
         If Dialog.ShowDialog(viewModel) Then ApplyChanges viewModel
17
18
         Set view = Nothing
19
20
     End Sub
21
     Private Sub view ApplyChanges(ByVal viewModel As ExampleModel)
22
23
         ApplyChanges viewModel
     End Sub
24
25
     Private Sub ApplyChanges(ByVal viewModel As ExampleModel)
26
27
         Sheet1.Range("A1").value = viewModel.field1
         Sheet1.Range("A2").value = viewModel.field2
28
29
     End Sub
```

So we have a Private WithEvents field that gets assigned in the Show method, and we handle the form's ApplyChanges event by invoking the ApplyChanges logic, which, for the sake of this example, takes the two fields and writes them to A1 and A2 on Sheet1; if you've read There is no worksheet (https://rubberduckvba.wordpress.com/2017/12/08/there-is-no-worksheet/) then you know how you can introduce an interface there to decouple the worksheet from the presenter, and then it doesn't matter if you're writing to a worksheet, a text file, or a database: the presenter doesn't need to know all the details.

The calling code in Module1 might look like this:

One problem here, is that the **View** implementation is coupled with the presenter (i.e. the presenter is creating the view): we need the concrete UserForm type in order for VBA to see the events; without further abstraction, we can't quite pass a <code>IDialogView</code> implementation to the presenter logic without popping up the actual dialog. Pieter Geerkens has a nice answer on Stack Overflow (https://stackoverflow.com/a/45825831/1188513) that describes how an **Adapter Pattern** can be used to solve this problem by introducing more interfaces, but covering this design pattern will be the subject of another article.





REPORT THIS AD

Posted in OOP, tutorials, vba Tagged design-patterns, oop, tutorial, vba



Published by Rubberduck VBA

I'm Mathieu Guindon (Microsoft MVP Office Apps & Services, 2018), you may have known me as "Mat's Mug" on Stack Overflow and Code Review Stack Exchange. I manage the Rubberduck open-source project, whose goal is to bring the Visual Basic Editor (VBE) - VBA's IDE - into the 21st century, by providing features modern IDE's provide. <u>View all posts by Rubberduck VBA</u>

11 thoughts on "Apply' logic for UserForm dialog"

1. **koitaki** *June* 10, 2018 <u>Reply</u>

Mille merci Mathieu, this is as usual *really* useful stuff.

I would love-love-love to use this MVP approach with Access. But haven't succeeded with it yet.

For example, with this particular case, when I try to use this approach on Access, it runs into specific difficulties.

Such as, a modal form seems to need to be opened via: DoCmd.OpenForm view.Name, acNormal, , , , acDialog

- 1) I don't know if that's actually opening the view form object)
- 2) This seems to create problems because where this line doesn't work so well:

If Dialog.ShowDialog(viewModel) Then ApplyChanges viewModel

Eg. upon closing the form, the code goes back to the Presenter class, and the form's this.IsCancelled loses its state

I'll keep working my way through these issues, and hopefully get there. But if in the meantime you have any ideas, I'd be happy to hear them \bigcirc

Cheers again, really appreciate what you're doing with this!

Chris

Rubberduck VBA June 10, 2018 Reply

Thanks! I'm not very familiar with Access, but I do know that Rubberduck sees Access forms are "document" modules, like a worksheet in Excel. This might explain it. Know that you can still have MSForms UserForm modules in an Access VBA project though: IIRC vanilla-AccessVBE hides the command, but Rubberduck's code explorer doesn't care and exposes the command anyway, so you can add a UserForm in Access if you need one. Hope it helps!

2. **Profex** September 21, 2018 Reply

I decided to convert my current project over to using proper OOP techniques, following your posts. It has a UserForm that has an Import button, Report Button and a Multi-Column-ListBox that lists all the projects from a database among other things. I'm having trouble deciding how far I want to go into defining the Model for the ListBox. It looks like I'm going to implement AddItem, List, Selected, ListCount & ListIndex properties. It's reminding me of my awful 64-bit COM wrapper for the .Net Listview control :(.

How would you deal with ListBox or ComboBox controls in the Model?

<u>Rubberduck VBA</u> <u>September 21, 2018</u> <u>Reply</u>

Controls themselves belong only in the view; in the model I'd have a getter that gives me the items I need to have in the listbox, and a read/write property for the current/selected item. The code-behind for the listbox would set the model's selecteditem on change, and the setter for the view's "model" property would populate the listbox. I hope it's not too confusing, it's a bit hard to be clear in a comment box.. it's very similar to a ViewModel in WPF/MVVM, except instead of XAML bindings you use the controls' event handlers to set the model properties – like I'm doing here with these textboxes.

1. **Profex** *November 2, 2018*

I ended up creating a model class that mimicked a lot of the the Listbox control properties/methods. If anyone (i.e. SmileyFtW) is interested in it, here is a link:

1	'@Folder("View.Model")
2	Option Explicit
3	op non Express
4	Private Const MODULE_NAME As String = "ListModel"
5	Titvate Collet Wie De EE_1 William File Starting Electrodes
6	Private Type TListModel
7	Data() As Variant
8	Selected() As Boolean
9	Columns As Integer
10	Count As Integer
11	Index As Integer
12	End Type
13	Ziia Type
14	Private this As TListModel
15	
16	Private Sub Class_Initialize()
17	this.Columns = 1
18	this.Index = -1
19	End Sub
20	
21	Public Sub Clear()
22	With this
23	Erase .Data
24	Erase .Selected
25	.Count = 0
26	'Columns = 0
27	.Index = -1
28	End With
29	End Sub
30	
31	Public Sub AddItem(Optional Item As Variant, Optional Index As Integer = -1)
32	Dim r As Integer, C As Integer
33	With this
34	If Index < -1 Or Index > .Count Then
35	Err.Raise 5, , "Invalid argument."
36	Else
37	ReDim Preserve .Data(.Columns - 1, .Count)
38	ReDim Preserve .Selected(.Count)
39	If $Index \ge 0$ Then
40	' Move all the data after the Index row, up one row.
41	For $r = .Count To Index + 1 Step -1$
42	For $C = 0$ To .Columns - 1
43	.Data(C, r) = .Data(C, r - 1)

82 With this 83 If Row < 0 Or Row >= .Count Then 84 Err.Raise 381, , "Could not get the List property. Invalid property-array rov 85 ElseIf Column < 0 Or Column >= .Columns Then 86 Err.Raise 381, , "Could not get the List property. Invalid property-array col-		
46 Next 7 Clear all the data in the Index row 48 For C = 0 To . Columns - 1 9 Set .Data(C, Index) = Nothing 50 Next 51 .Selected(Index) = False 52 Else 'Set the Index to the next row 53	44	Next
Clear all the data in the Index row	45	.Selected(r) = .Selected(r - 1)
48 For C = 0 To .Columns - 1 49 Set .Data(C, Index) = Nothing 50 Next 51 .Selected(Index) = False 52 Else 'Set the Index to the next row 53	46	Next
Set .Data(C, Index) = Nothing	47	' Clear all the data in the Index row
50 Next 51 Selected(Index) = False 52 Else 'Set the Index to the next row 53 Index = .Count 54 End If 55 If Not IsMissing(Item) Then .Data(0, Index) = Item 56 .Count = .Count + 1 57 End If 58 End With 59 End Sub 60 61 Public Sub Removeltem(Index As Integer) 62 Dim r As Integer, C As Integer 63 With this 64 If Index < 0 Or Index >= .Count Then 65 Err.Raise 5, , "Invalid argument." 66 Else 67 'Move all the data after the Index row, up one row. 68 For r = Index + 1 To .Count - 1 69 For C = 0 To .Columns - 1 70 .Data(C, r - 1) = .Data(C, r) 71 Next 72 .Selected(r - 1) = .Selected(r) 73 Next 74 .Count = .Count - 1 75 ReDim Preserve .Data(.Columns - 1, .Count - 1) 76 ReDim Preserve .Selected(.Count - 1) 77 End If 78 End With 79 End Sub 80 81 Public Property Get List(Row As Integer, Optional Column As Integer = 0) As Variand Substitution of the Column in the Colum	48	For C = 0 To .Columns - 1
51	49	Set .Data(C, Index) = Nothing
Else 'Set the Index to the next row	50	Next
Index = .Count	51	.Selected(Index) = False
54 End If 55 If Not IsMissing(Item) Then .Data(0, Index) = Item 56Count = .Count + 1 57 End If 58 End With 59 End Sub 60 61 Public Sub Removeltem(Index As Integer) 62 Dim r As Integer, C As Integer 63 With this 64 If Index < 0 Or Index >= .Count Then 65 Err.Raise 5, , "Invalid argument." 66 Else 67 'Move all the data after the Index row, up one row. 68 For r = Index + 1 To .Count - 1 69 For C = 0 To .Columns - 1 70Data(C, r - 1) = .Data(C, r) 71 Next 72Selected(r - 1) = .Selected(r) 73 Next 74Count = .Count - 1 75 ReDim Preserve .Data(.Columns - 1, .Count - 1) 76 ReDim Preserve .Bata(.Columns - 1, .Count - 1) 77 End If 78 End With 79 End Sub 80 81 Public Property Get List(Row As Integer, Optional Column As Integer = 0) As Variable Subset Columns = .Column >= .Columns = .Colum	52	Else 'Set the Index to the next row
55 If Not IsMissing(Item) Then .Data(0, Index) = Item .Count = .Count + 1 57 End If 58 End With 59 End Sub 60 61 Public Sub Removeltem(Index As Integer) 62 Dim r As Integer, C As Integer 63 With this 64 If Index < 0 Or Index >= .Count Then 65 Er.Raise 5, , "Invalid argument." 66 Else 67 'Move all the data after the Index row, up one row. 68 For r = Index + 1 To .Count - 1 69 For C = 0 To .Columns - 1 70 .Data(C, r - 1) = .Data(C, r) 71 Next 72 .Selected(r - 1) = .Selected(r) 73 Next 74 .Count = .Count - 1 75 ReDim Preserve .Data(.Columns - 1, .Count - 1) 76 ReDim Preserve .Selected(.Count - 1) 77 End If 78 End With 79 End Sub 80 81 Public Property Get List(Row As Integer, Optional Column As Integer = 0) As Variation of the column of	53	Index = .Count
56	54	End If
57 End If 58 End With 59 End Sub 60 61 Public Sub Removeltem(Index As Integer) 62 Dim r As Integer, C As Integer 63 With this 64 If Index < 0 Or Index >= .Count Then 65 Err.Raise 5, , "Invalid argument." 66 Else 67 'Move all the data after the Index row, up one row. 68 For r = Index + 1 To .Count - 1 69 For C = 0 To .Columns - 1 70 .Data(C, r - 1) = .Data(C, r) 71 Next 72 .Selected(r - 1) = .Selected(r) 73 Next 74 .Count = .Count - 1 75 ReDim Preserve .Data(.Columns - 1, .Count - 1) 76 ReDim Preserve .Selected(.Count - 1) 77 End If 78 End With 79 End Sub 80 81 Public Property Get List(Row As Integer, Optional Column As Integer = 0) As Variable Subset of the Column of the List property. Invalid property-array row Elself Column < 0 Or Column >= .Columns Then 86 Err.Raise 381, , "Could not get the List property. Invalid property-array column of the Column of the List property. Invalid property-array column of the Column of the List property. Invalid property-array column of the Column of the List property. Invalid property-array column of the Column of the Column of the List property. Invalid property-array column of the Column of the List property. Invalid property-array column of the Column of the Column of the List property. Invalid property-array column of the Column of the Column of the Column of the List property. Invalid property-array column of the Column of t	55	If Not IsMissing(Item) Then .Data(0, Index) = Item
58 End With 59 End Sub 60 61 Public Sub RemoveItem(Index As Integer) 62 Dim r As Integer, C As Integer 63 With this 64 If Index < 0 Or Index >= .Count Then 65 Err.Raise 5, , "Invalid argument." 66 Else 67 'Move all the data after the Index row, up one row. 68 For r = Index + 1 To .Count - 1 69 For C = 0 To .Columns - 1 70 .Data(C, r - 1) = .Data(C, r) 71 Next 72 .Selected(r - 1) = .Selected(r) 73 Next 74 .Count = .Count - 1 75 ReDim Preserve .Data(.Columns - 1, .Count - 1) 76 ReDim Preserve .Selected(.Count - 1) 77 End If 78 End With 79 End Sub 80 81 Public Property Get List(Row As Integer, Optional Column As Integer = 0) As Variable Sub 81 If Row < 0 Or Row >= .Count Then 84 Err.Raise 381, , "Could not get the List property. Invalid property-array row 85 Elself Column < 0 Or Column >= .Columns Then 86 Err.Raise 381, , "Could not get the List property. Invalid property-array columns = .Columns Then	56	.Count = .Count + 1
59 End Sub 60 61 Public Sub RemoveItem(Index As Integer) 62 Dim r As Integer, C As Integer 63 With this 64 If Index < 0 Or Index >= .Count Then 65 Err.Raise 5, , "Invalid argument." 66 Else 67 'Move all the data after the Index row, up one row. 68 For r = Index + 1 To .Count - 1 69 For C = 0 To .Columns - 1 70 .Data(C, r - 1) = .Data(C, r) 71 Next 72 .Selected(r - 1) = .Selected(r) 73 Next 74 .Count = .Count - 1 75 ReDim Preserve .Data(.Columns - 1, .Count - 1) 76 ReDim Preserve .Selected(.Count - 1) 77 End If 78 End With 79 End Sub 80 81 Public Property Get List(Row As Integer, Optional Column As Integer = 0) As Variable With this 83 If Row < 0 Or Row >= .Count Then 84 Err.Raise 381, , "Could not get the List property. Invalid property-array row 85 Elself Column < 0 Or Column >= .Columns Then 86 Err.Raise 381, , "Could not get the List property. Invalid property-array col	57	End If
60 61 Public Sub RemoveItem(Index As Integer) 62 Dim r As Integer, C As Integer 63 With this 64 If Index < 0 Or Index >= .Count Then 65 Err.Raise 5, , "Invalid argument." 66 Else 67 'Move all the data after the Index row, up one row. 68 For r = Index + 1 To .Count - 1 69 For C = 0 To .Columns - 1 70 .Data(C, r - 1) = .Data(C, r) 71 Next 72 .Selected(r - 1) = .Selected(r) 73 Next 74 .Count = .Count - 1 75 ReDim Preserve .Data(.Columns - 1, .Count - 1) 76 ReDim Preserve .Selected(.Count - 1) 77 End If 78 End With 79 End Sub 80 81 Public Property Get List(Row As Integer, Optional Column As Integer = 0) As Variable With this 83 If Row < 0 Or Row >= .Count Then 84 Err.Raise 381, , "Could not get the List property. Invalid property-array row 85 Elself Column < 0 Or Column >= .Columns Then 86 Err.Raise 381, , "Could not get the List property. Invalid property-array col	58	End With
61 Public Sub RemoveItem(Index As Integer) 62 Dim r As Integer, C As Integer 63 With this 64 If Index < 0 Or Index >= .Count Then 65 Err.Raise 5, , "Invalid argument." 66 Else 67 'Move all the data after the Index row, up one row. 68 For r = Index + 1 To .Count - 1 69 For C = 0 To .Columns - 1 70 .Data(C, r - 1) = .Data(C, r) 71 Next 72 .Selected(r - 1) = .Selected(r) 73 Next 74 .Count = .Count - 1 75 ReDim Preserve .Data(.Columns - 1, .Count - 1) 76 ReDim Preserve .Selected(.Count - 1) 77 End If 78 End With 79 End Sub 80 81 Public Property Get List(Row As Integer, Optional Column As Integer = 0) As Variable Sub	59	End Sub
62 Dim r As Integer, C As Integer 63 With this 64 If Index < 0 Or Index >= .Count Then 65 Err.Raise 5, , "Invalid argument." 66 Else 67 'Move all the data after the Index row, up one row. 68 For r = Index + 1 To .Count - 1 69 For C = 0 To .Columns - 1 70 .Data(C, r - 1) = .Data(C, r) 71 Next 72 .Selected(r - 1) = .Selected(r) 73 Next 74 .Count = .Count - 1 75 ReDim Preserve .Data(.Columns - 1, .Count - 1) 76 ReDim Preserve .Selected(.Count - 1) 77 End If 78 End With 79 End Sub 80 81 Public Property Get List(Row As Integer, Optional Column As Integer = 0) As Variable 1 82 With this 83 If Row < 0 Or Row >= .Count Then 84 Err.Raise 381, , "Could not get the List property. Invalid property-array row 85 Elself Column < 0 Or Column >= .Columns Then 86 Err.Raise 381, , "Could not get the List property. Invalid property-array columns = .Columns Then	60	
63 With this 64 If Index < 0 Or Index >= .Count Then 65 Err.Raise 5, , "Invalid argument." 66 Else 67 'Move all the data after the Index row, up one row. 68 For r = Index + 1 To .Count - 1 69 For C = 0 To .Columns - 1 70 .Data(C, r - 1) = .Data(C, r) 71 Next 72 .Selected(r - 1) = .Selected(r) 73 Next 74 .Count = .Count - 1 75 ReDim Preserve .Data(.Columns - 1, .Count - 1) 76 ReDim Preserve .Selected(.Count - 1) 77 End If 78 End With 79 End Sub 80 81 Public Property Get List(Row As Integer, Optional Column As Integer = 0) As Variable With this 83 If Row < 0 Or Row >= .Count Then 84 Err.Raise 381, , "Could not get the List property. Invalid property-array row 85 Elself Column < 0 Or Column >= .Columns Then 86 Err.Raise 381, , "Could not get the List property. Invalid property-array columns and the column of the column in the column of the column in the column of the column in the column	61	Public Sub RemoveItem(Index As Integer)
64 If Index < 0 Or Index >= .Count Then 65 Err.Raise 5, , "Invalid argument." 66 Else 67 'Move all the data after the Index row, up one row. 68 For r = Index + 1 To .Count - 1 69 For C = 0 To .Columns - 1 70 .Data(C, r - 1) = .Data(C, r) 71 Next 72 .Selected(r - 1) = .Selected(r) 73 Next 74 .Count = .Count - 1 75 ReDim Preserve .Data(.Columns - 1, .Count - 1) 76 ReDim Preserve .Selected(.Count - 1) 77 End If 78 End With 79 End Sub 80 81 Public Property Get List(Row As Integer, Optional Column As Integer = 0) As Variable Sub 81 If Row < 0 Or Row >= .Count Then 84 Err.Raise 381, , "Could not get the List property. Invalid property-array row 85 Elself Column < 0 Or Column >= .Columns Then 86 Err.Raise 381, , "Could not get the List property. Invalid property-array columns Substituted in the serres of the serres	62	Dim r As Integer, C As Integer
65 Err.Raise 5, , "Invalid argument." 66 Else 67 'Move all the data after the Index row, up one row. 68 For r = Index + 1 To .Count - 1 69 For C = 0 To .Columns - 1 70 .Data(C, r - 1) = .Data(C, r) 71 Next 72 .Selected(r - 1) = .Selected(r) 73 Next 74 .Count = .Count - 1 75 ReDim Preserve .Data(.Columns - 1, .Count - 1) 76 ReDim Preserve .Selected(.Count - 1) 77 End If 78 End With 79 End Sub 80 81 Public Property Get List(Row As Integer, Optional Column As Integer = 0) As Variable With this 82 With this 83 If Row < 0 Or Row >= .Count Then 84 Err.Raise 381, , "Could not get the List property. Invalid property-array row 85 ElseIf Column < 0 Or Column >= .Columns Then 86 Err.Raise 381, , "Could not get the List property. Invalid property-array columns and the column in the column of the column in the colum	63	With this
66 Else 67 'Move all the data after the Index row, up one row. 68 For r = Index + 1 To .Count - 1 69 For C = 0 To .Columns - 1 70 .Data(C, r - 1) = .Data(C, r) 71 Next 72 .Selected(r - 1) = .Selected(r) 73 Next 74 .Count = .Count - 1 75 ReDim Preserve .Data(.Columns - 1, .Count - 1) 76 ReDim Preserve .Selected(.Count - 1) 77 End If 78 End With 79 End Sub 80 81 Public Property Get List(Row As Integer, Optional Column As Integer = 0) As Variable With this 83 If Row < 0 Or Row >= .Count Then 84 Err.Raise 381, , "Could not get the List property. Invalid property-array row 85 Elself Column < 0 Or Column >= .Columns Then 86 Err.Raise 381, , "Could not get the List property. Invalid property-array columns Then	64	If Index < 0 Or Index >= .Count Then
'Move all the data after the Index row, up one row. For r = Index + 1 To .Count - 1 For C = 0 To .Columns - 1 Data(C, r - 1) = .Data(C, r) Next Selected(r - 1) = .Selected(r) Next Count = .Count - 1 ReDim Preserve .Data(.Columns - 1, .Count - 1) ReDim Preserve .Selected(.Count - 1) End If Bend With Find Sub With this With this If Row < 0 Or Row >= .Count Then Selected Columns - 1	65	Err.Raise 5, , "Invalid argument."
68 For r = Index + 1 To .Count - 1 69 For C = 0 To .Columns - 1 70 .Data(C, r - 1) = .Data(C, r) 71 Next 72 .Selected(r - 1) = .Selected(r) 73 Next 74 .Count = .Count - 1 75 ReDim Preserve .Data(.Columns - 1, .Count - 1) 76 ReDim Preserve .Selected(.Count - 1) 77 End If 78 End With 79 End Sub 80 81 Public Property Get List(Row As Integer, Optional Column As Integer = 0) As Variable With this 83 If Row < 0 Or Row >= .Count Then 84 Err.Raise 381, , "Could not get the List property. Invalid property-array row 85 Elself Column < 0 Or Column >= .Columns Then 86 Err.Raise 381, , "Could not get the List property. Invalid property-array columns	66	Else
For C = 0 To .Columns - 1 Data(C, r - 1) = .Data(C, r) Next Selected(r - 1) = .Selected(r) Next Count = .Count - 1 ReDim Preserve .Data(.Columns - 1, .Count - 1) ReDim Preserve .Selected(.Count - 1) End If End With Public Property Get List(Row As Integer, Optional Column As Integer = 0) As Variable With this If Row < 0 Or Row >= .Count Then Err.Raise 381, , "Could not get the List property. Invalid property-array row Elself Column < 0 Or Column >= .Columns Then Err.Raise 381, , "Could not get the List property. Invalid property-array columns are columns as a column of the column of the column of the columns are columns."	67	' Move all the data after the Index row, up one row.
70	68	For $r = Index + 1$ To .Count - 1
71 Next 72 .Selected(r - 1) = .Selected(r) 73 Next 74 .Count = .Count - 1 75 ReDim Preserve .Data(.Columns - 1, .Count - 1) 76 ReDim Preserve .Selected(.Count - 1) 77 End If 78 End With 79 End Sub 80 81 Public Property Get List(Row As Integer, Optional Column As Integer = 0) As Variable With this 82 With this 83 If Row < 0 Or Row >= .Count Then 84 Err.Raise 381, , "Could not get the List property. Invalid property-array row 85 Elself Column < 0 Or Column >= .Columns Then 86 Err.Raise 381, , "Could not get the List property. Invalid property-array columns Then	69	For C = 0 To .Columns - 1
72 .Selected(r - 1) = .Selected(r) 73 Next 74 .Count = .Count - 1 75 ReDim Preserve .Data(.Columns - 1, .Count - 1) 76 ReDim Preserve .Selected(.Count - 1) 77 End If 78 End With 79 End Sub 80 81 81 Public Property Get List(Row As Integer, Optional Column As Integer = 0) As Variant States 82 With this 83 If Row < 0 Or Row >= .Count Then 84 Err.Raise 381, , "Could not get the List property. Invalid property-array row 85 ElseIf Column < 0 Or Column >= .Columns Then 86 Err.Raise 381, , "Could not get the List property. Invalid property-array columns	70	.Data(C, r - 1) = .Data(C, r)
Next Count = .Count - 1 ReDim Preserve .Data(.Columns - 1, .Count - 1) ReDim Preserve .Selected(.Count - 1) ReDim Preserve	71	Next
74	72	.Selected(r - 1) = .Selected(r)
75 ReDim Preserve .Data(.Columns - 1, .Count - 1) 76 ReDim Preserve .Selected(.Count - 1) 77 End If 78 End With 79 End Sub 80 81 Public Property Get List(Row As Integer, Optional Column As Integer = 0) As Variable With this 82 With this 83 If Row < 0 Or Row >= .Count Then 84 Err.Raise 381, , "Could not get the List property. Invalid property-array row 85 Elself Column < 0 Or Column >= .Columns Then 86 Err.Raise 381, , "Could not get the List property. Invalid property-array columns and the column in the	73	Next
76 ReDim Preserve .Selected(.Count - 1) 77 End If 78 End With 79 End Sub 80 81 Public Property Get List(Row As Integer, Optional Column As Integer = 0) As Variable With this 82 With this 83 If Row < 0 Or Row >= .Count Then 84 Err.Raise 381, , "Could not get the List property. Invalid property-array row 85 ElseIf Column < 0 Or Column >= .Columns Then 86 Err.Raise 381, , "Could not get the List property. Invalid property-array columns are columns."	74	.Count = .Count - 1
77 End If 78 End With 79 End Sub 80 81 Public Property Get List(Row As Integer, Optional Column As Integer = 0) As Variable With this 82 With this 83 If Row < 0 Or Row >= .Count Then 84 Err.Raise 381, , "Could not get the List property. Invalid property-array row 85 ElseIf Column < 0 Or Column >= .Columns Then 86 Err.Raise 381, , "Could not get the List property. Invalid property-array columns Then	75	ReDim Preserve .Data(.Columns - 1, .Count - 1)
 End With End Sub Public Property Get List(Row As Integer, Optional Column As Integer = 0) As Variance With this If Row < 0 Or Row >= .Count Then Err.Raise 381, , "Could not get the List property. Invalid property-array row ElseIf Column < 0 Or Column >= .Columns Then Err.Raise 381, , "Could not get the List property. Invalid property-array columns Err.Raise 381, , "Could not get the List property. Invalid property-array columns 	76	ReDim Preserve .Selected(.Count - 1)
79 End Sub 80 81 Public Property Get List(Row As Integer, Optional Column As Integer = 0) As Variable With this 82 With this 83 If Row < 0 Or Row >= .Count Then 84 Err.Raise 381, , "Could not get the List property. Invalid property-array row 85 ElseIf Column < 0 Or Column >= .Columns Then 86 Err.Raise 381, , "Could not get the List property. Invalid property-array columns Then	77	End If
80 81 Public Property Get List(Row As Integer, Optional Column As Integer = 0) As Variable With this 82 With this 83 If Row < 0 Or Row >= .Count Then 84 Err.Raise 381, , "Could not get the List property. Invalid property-array row 85 ElseIf Column < 0 Or Column >= .Columns Then 86 Err.Raise 381, , "Could not get the List property. Invalid property-array columns Then	78	End With
Public Property Get List(Row As Integer, Optional Column As Integer = 0) As Variable With this If Row < 0 Or Row >= .Count Then Err.Raise 381, , "Could not get the List property. Invalid property-array row ElseIf Column < 0 Or Column >= .Columns Then Err.Raise 381, , "Could not get the List property. Invalid property-array columns Then	79	End Sub
 With this If Row < 0 Or Row >= .Count Then Err.Raise 381, , "Could not get the List property. Invalid property-array row ElseIf Column < 0 Or Column >= .Columns Then Err.Raise 381, , "Could not get the List property. Invalid property-array col- 	80	
If Row < 0 Or Row >= .Count Then Err.Raise 381, , "Could not get the List property. Invalid property-array row ElseIf Column < 0 Or Column >= .Columns Then Err.Raise 381, , "Could not get the List property. Invalid property-array col-	81	Public Property Get List(Row As Integer, Optional Column As Integer = 0) As Varia
84 Err.Raise 381, , "Could not get the List property. Invalid property-array rov 85 ElseIf Column < 0 Or Column >= .Columns Then 86 Err.Raise 381, , "Could not get the List property. Invalid property-array col-	82	With this
85 ElseIf Column < 0 Or Column >= .Columns Then 86 Err.Raise 381, , "Could not get the List property. Invalid property-array col-	83	If Row < 0 Or Row >= .Count Then
86 Err.Raise 381, , "Could not get the List property. Invalid property-array col-	84	Err Daiga 201 "Could not get the List property. Invalid property array revy
	85	Eff. Raise 361, Could not get the List property. Invalid property-array fow
87 Else		
U. Libe		
88 List = .Data(Column, Row)		ElseIf Column < 0 Or Column >= .Columns Then

89	End If
90	End With
91	End Property
92	
93	Public Property Let List(Row As Integer, Column As Integer, Value As Variant)
94	With this
95	If Row < 0 Or Row >= .Count Then
96	Err.Raise 381, , "Could not get the List property. Invalid property-array row i
97	ElseIf Column < 0 Or Column >= .Columns Then
98	Err.Raise 381, , "Could not get the List property. Invalid property-array colur
99	Else
100	.Data(Column, Row) = Value
101	End If
102	End With
103	End Property
104	
105	Public Property Get Selected(Index As Integer) As Boolean
106	With this
107	If Index < 0 Or Index >= .Count Then
108	Err.Raise 381, , "Could not get the List property. Invalid property-array index
109	Else
110	Selected = .Selected(Index)
111	End If
112	End With
113	End Property
114	
115	Public Property Let Selected(Index As Integer, Value As Boolean)
116	With this
117	If Index < 0 Or Index >= .Count Then
118	Err.Raise 381, , "Could not get the List property. Invalid property-array index
119	Else
120	.Selected(Index) = Value
121	End If
122	End With
123	End Property
124	
125	Public Property Get ListCount() As Integer
126	ListCount = this.Count
127	End Property
128	
129	Public Property Get ListIndex() As Integer
130	ListIndex = this.Index
131	End Property
132	
133	Public Property Let ListIndex(Value As Integer)

135 If Value <-1 Or Value >= .Count Then 136 Err.Raise 5, , "Invalid argument." 137 Else 138 .Index = Value 139 End If 140 End With 141 End Property 142 143 Public Property Get ColumnCount() As Integer 144 ColumnCount = this.Columns 145 End Property 146 147 Public Property Let ColumnCount(Value As Integer) 148 Dim NewData() As Variant 149 Dim r As Integer, C As Integer 150 With this 151 If Value <= 0 Then 152 Err.Raise 5, , "Invalid argument." 153 Else 154 If .Count > 0 And .Columns <> Value Then	134	With this
137 Else 138Index = Value 139 End If 140 End With 141 End Property 142 143 Public Property Get ColumnCount() As Integer 144 ColumnCount = this.Columns 145 End Property 146 147 Public Property Let ColumnCount(Value As Integer) 148 Dim NewData() As Variant 149 Dim r As Integer, C As Integer 150 With this 151 If Value <= 0 Then 152 Err.Raise 5, , "Invalid argument." 153 Else 154 If .Count > 0 And .Columns ◇ Value Then 155 If the columns change, we can't redim the array, we need to create a new I 156 ReDim NewData(Value - 1, .Count - 1) 157 For r = 0 To .Count - 1 158 For C = 0 To Value - 1 159 NewData(C, r) = .Data(C, r) 160 Next 161 Next 162Data = NewData 163 Erase NewData 164 End If 165Columns = Value 166 End If 167 End With		
137 Else 138Index = Value 139 End If 140 End With 141 End Property 142 143 Public Property Get ColumnCount() As Integer 144 ColumnCount = this.Columns 145 End Property 146 147 Public Property Let ColumnCount(Value As Integer) 148 Dim NewData() As Variant 149 Dim r As Integer, C As Integer 150 With this 151 If Value <= 0 Then 152 Err.Raise 5, , "Invalid argument." 153 Else 154 If .Count > 0 And .Columns <> Value Then 155 'If the columns change, we can't redim the array, we need to create a new I 156 ReDim NewData(Value - 1, .Count - 1) 157 For r = 0 To .Count - 1 158 For C = 0 To Value - 1 159 NewData(C, r) = .Data(C, r) 160 Next 161 Next 162Data = NewData 163 Erase NewData 164 End If 165Columns = Value 166 End If 167 End With	136	Err.Raise 5, , "Invalid argument."
139End If140End With141End Property142143Public Property Get ColumnCount() As Integer144ColumnCount = this.Columns145End Property146147Public Property Let ColumnCount(Value As Integer)148Dim NewData() As Variant149Dim r As Integer, C As Integer150With this151If Value <= 0 Then	137	<u> </u>
140 End With 141 End Property 142 143 Public Property Get ColumnCount() As Integer 144 ColumnCount = this.Columns 145 End Property 146 147 Public Property Let ColumnCount(Value As Integer) 148 Dim NewData() As Variant 149 Dim r As Integer, C As Integer 150 With this 151 If Value <= 0 Then 152 Err.Raise 5, , "Invalid argument." 153 Else 154 If .Count > 0 And .Columns ⇒ Value Then 155 'If the columns change, we can't redim the array, we need to create a new I 156 ReDim NewData(Value - 1, .Count - 1) 157 For r = 0 To .Count - 1 158 For C = 0 To Value - 1 159 NewData(C, r) = .Data(C, r) 160 Next 161 Next 162 .Data = NewData 163 Erase NewData 164 End If 165 .Columns = Value 166 End If 167 End With	138	.Index = Value
141 End Property 142 143 Public Property Get ColumnCount() As Integer 144 ColumnCount = this.Columns 145 End Property 146 147 Public Property Let ColumnCount(Value As Integer) 148 Dim NewData() As Variant 149 Dim r As Integer, C As Integer 150 With this 151 If Value <= 0 Then 152 Err.Raise 5, , "Invalid argument." 153 Else 154 If .Count > 0 And .Columns ⇔ Value Then 155 'If the columns change, we can't redim the array, we need to create a new I 156 ReDim NewData(Value - 1, .Count - 1) 157 For r = 0 To .Count - 1 158 For C = 0 To Value - 1 159 NewData(C, r) = .Data(C, r) 160 Next 161 Next 162 .Data = NewData 163 Erase NewData 164 End If 165 .Columns = Value 166 End If 167 End With	139	End If
142 143 Public Property Get ColumnCount() As Integer 144 ColumnCount = this.Columns 145 End Property 146 147 Public Property Let ColumnCount(Value As Integer) 148 Dim NewData() As Variant 149 Dim r As Integer, C As Integer 150 With this 151 If Value <= 0 Then 152 Err.Raise 5, , "Invalid argument." 153 Else 154 If .Count > 0 And .Columns ⇔ Value Then 155 'If the columns change, we can't redim the array, we need to create a new I 156 ReDim NewData(Value - 1, .Count - 1) 157 For r = 0 To .Count - 1 158 For C = 0 To Value - 1 159 NewData(C, r) = .Data(C, r) 160 Next 161 Next 162 .Data = NewData 163 Erase NewData 164 End If 165 .Columns = Value 166 End If 167 End With	140	End With
143 Public Property Get ColumnCount() As Integer 144 ColumnCount = this.Columns 145 End Property 146 Public Property Let ColumnCount(Value As Integer) 148 Dim NewData() As Variant 149 Dim r As Integer, C As Integer 150 With this 151 If Value <= 0 Then	141	End Property
144 ColumnCount = this.Columns 145 End Property 146 147 Public Property Let ColumnCount(Value As Integer) 148 Dim NewData() As Variant 149 Dim r As Integer, C As Integer 150 With this 151 If Value <= 0 Then 152 Err.Raise 5, , "Invalid argument." 153 Else 154 If .Count > 0 And .Columns ⇔ Value Then 155 'If the columns change, we can't redim the array, we need to create a new I 156 ReDim NewData(Value - 1, .Count - 1) 157 For r = 0 To .Count - 1 158 For C = 0 To Value - 1 159 NewData(C, r) = .Data(C, r) 160 Next 161 Next 162 .Data = NewData 163 Erase NewData 164 End If 165 .Columns = Value 166 End If 167 End With	142	
145End Property146147Public Property Let ColumnCount(Value As Integer)148Dim NewData() As Variant149Dim r As Integer, C As Integer150With this151If Value <= 0 Then	143	Public Property Get ColumnCount() As Integer
146147Public Property Let ColumnCount(Value As Integer)148Dim NewData() As Variant149Dim r As Integer, C As Integer150With this151If Value <= 0 Then	144	ColumnCount = this.Columns
147Public Property Let ColumnCount(Value As Integer)148Dim NewData() As Variant149Dim r As Integer, C As Integer150With this151If Value <= 0 Then	145	End Property
148Dim NewData() As Variant149Dim r As Integer, C As Integer150With this151If Value <= 0 Then	146	
149 Dim r As Integer, C As Integer 150 With this 151 If Value <= 0 Then 152 Err.Raise 5, , "Invalid argument." 153 Else 154 If .Count > 0 And .Columns ❖ Value Then 155 'If the columns change, we can't redim the array, we need to create a new I 156 ReDim NewData(Value - 1, .Count - 1) 157 For r = 0 To .Count - 1 158 For C = 0 To Value - 1 159 NewData(C, r) = .Data(C, r) 160 Next 161 Next 162 .Data = NewData 163 Erase NewData 164 End If 165 .Columns = Value 166 End With	147	Public Property Let ColumnCount(Value As Integer)
150 With this 151 If Value <= 0 Then 152 Err.Raise 5, , "Invalid argument." 153 Else 154 If .Count > 0 And .Columns ❖ Value Then 155 'If the columns change, we can't redim the array, we need to create a new I 156 ReDim NewData(Value - 1, .Count - 1) 157 For r = 0 To .Count - 1 158 For C = 0 To Value - 1 159 NewData(C, r) = .Data(C, r) 160 Next 161 Next 162 .Data = NewData 163 Erase NewData 164 End If 165 .Columns = Value 166 End If 167 End With	148	Dim NewData() As Variant
151 If Value <= 0 Then 152 Err.Raise 5, , "Invalid argument." 153 Else 154 If .Count > 0 And .Columns ❖ Value Then 155 'If the columns change, we can't redim the array, we need to create a new I 156 ReDim NewData(Value - 1, .Count - 1) 157 For r = 0 To .Count - 1 158 For C = 0 To Value - 1 159 NewData(C, r) = .Data(C, r) 160 Next 161 Next 162 .Data = NewData 163 Erase NewData 164 End If 165 .Columns = Value 166 End If 167 End With	149	Dim r As Integer, C As Integer
152 Err.Raise 5, , "Invalid argument." 153 Else 154 If .Count > 0 And .Columns Value Then 155 'If the columns change, we can't redim the array, we need to create a new I 156 ReDim NewData(Value - 1, .Count - 1) 157 For r = 0 To .Count - 1 158 For C = 0 To Value - 1 159 NewData(C, r) = .Data(C, r) 160 Next 161 Next 162 .Data = NewData 163 Erase NewData 164 End If 165 .Columns = Value 166 End If 167 End With	150	With this
153 Else 154 If .Count > 0 And .Columns Value Then 155 'If the columns change, we can't redim the array, we need to create a new I 156 ReDim NewData(Value - 1, .Count - 1) 157 For r = 0 To .Count - 1 158 For C = 0 To Value - 1 159 NewData(C, r) = .Data(C, r) 160 Next 161 Next 162 .Data = NewData 163 Erase NewData 164 End If 165 .Columns = Value 166 End If 167 End With	151	If Value <= 0 Then
154 If .Count > 0 And .Columns Value Then 155 'If the columns change, we can't redim the array, we need to create a new I 156 ReDim NewData(Value - 1, .Count - 1) 157 For r = 0 To .Count - 1 158 For C = 0 To Value - 1 159 NewData(C, r) = .Data(C, r) 160 Next 161 Next 162 .Data = NewData 163 Erase NewData 164 End If 165 .Columns = Value 166 End If 167 End With	152	Err.Raise 5, , "Invalid argument."
'If the columns change, we can't redim the array, we need to create a new I ReDim NewData(Value - 1, .Count - 1) For r = 0 To .Count - 1 For C = 0 To Value - 1 NewData(C, r) = .Data(C, r) Next Next Loata = NewData Erase NewData Erase NewData Columns = Value End If End With	153	Else
156 ReDim NewData(Value - 1, .Count - 1) 157 For r = 0 To .Count - 1 158 For C = 0 To Value - 1 159 NewData(C, r) = .Data(C, r) 160 Next 161 Next 162 .Data = NewData 163 Erase NewData 164 End If 165 .Columns = Value 166 End If 167 End With	154	If .Count > 0 And .Columns ⇔ Value Then
157 For r = 0 To .Count - 1 158 For C = 0 To Value - 1 159 NewData(C, r) = .Data(C, r) 160 Next 161 Next 162 .Data = NewData 163 Erase NewData 164 End If 165 .Columns = Value 166 End If 167 End With	155	' If the columns change, we can't redim the array, we need to create a new I
158 For C = 0 To Value - 1 159 NewData(C, r) = .Data(C, r) 160 Next 161 Next 162 .Data = NewData 163 Erase NewData 164 End If 165 .Columns = Value 166 End If 167 End With	156	ReDim NewData(Value - 1, .Count - 1)
159 NewData(C, r) = .Data(C, r) 160 Next 161 Next 162 .Data = NewData 163 Erase NewData 164 End If 165 .Columns = Value 166 End If 167 End With	157	For $r = 0$ To .Count - 1
160 Next 161 Next 162 .Data = NewData 163 Erase NewData 164 End If 165 .Columns = Value 166 End If 167 End With	158	For C = 0 To Value - 1
161 Next 162 .Data = NewData 163 Erase NewData 164 End If 165 .Columns = Value 166 End If 167 End With	159	NewData(C, r) = .Data(C, r)
162 .Data = NewData 163 Erase NewData 164 End If 165 .Columns = Value 166 End If 167 End With	160	Next
163 Erase NewData 164 End If 165 .Columns = Value 166 End If 167 End With	161	Next
164 End If 165 .Columns = Value 166 End If 167 End With	-	
165 .Columns = Value 166 End If 167 End With		
166 End If 167 End With		
167 End With	-	
	-	
168 End Property		
	168	End Property

view raw gistfile1.txt hosted with ♥ by GitHub

In the form, I called the following DisplayProjectList routine whenever I needed to update Listbox (Activate/Add/Remove):

Private Sub DisplayProjectList() Dim i As Integer, j As Integer lstProjects.Clear With this.Model.ProjectList For i = 0 To .ListCount – 1 lstProjects.AddItem .List(i, 0) For j = 1 To .ColumnCount – 1
lstProjects.List(i, j) = .List(i, j)
Next
lstProjects.Selected(i) = .Selected(i)
Next
lstProjects.ListIndex = .ListIndex
'If lstProjects.ListCount > 0 And lstProjects.ListIndex < 0 Then lstProjects.ListIndex = 0
End With
End Sub

Note: I ended up not caring about what was selected on Add (Remove doesn't matter), but because the Model for the ListBox included a Selected property, I also had the following:

Private Sub lstProjects_Change()
Dim i As Integer
With this.Model.ProjectList
For i = 0 To lstProjects.ListCount – 1
.Selected(i) = lstProjects.Selected(i)
Next
End With
End Sub

For the CobmoBox control, or if you just don't care about preserving the selected items when adding a new item, you would remove all references to the Selected properties.

3. SmileyFtW October 24, 2018 Reply

Great stuff. If there were to be an option on the form to show one or more subsets of the list presented in the combo box how might that be done? Say there was a list of 10 values in the data table and the user could select to choose to show only the even ones, the odd ones or all via check boxes or similar, what might that be implemented?

Rubberduck VBA October 24, 2018 Reply

There isn't One True Way, but I think I'd try to keep the presenter responsible for knowing what to do to get the [filtered] data (worksheet, db, hard-coded, whatever), and overwrite a property in the model that, when assigned, raises an event that the view can handle. Or the view could have some 'Refresh' method that clears & re-populates the comboboxes from the updated model contents; view.Refresh would be invoked from the presenter, after it finishes updating the model. So the model needs not only a property for the user-selected combobox value, it also needs a property for the desired filter, and then a property for the available values as per that filter. Basically you just do whatever needs to be done to systematically defer work out of the view and into the presenter. Eventually the model grows too large and confusing, so you keep the "model" stuff (i.e. selected values, user inputs) there and pull the "view model" stuff (i.e. filters, combobox/listbox sources) out into a new class to keep things manageable & clean.

1. **Profex** November 2, 2018 Reply

I posted the code that I used for the basic ListBox above. It includes a Selected property in the model, so you can modify the selected item from either the view or the model. I didn't get to the point where I needed the view.Refresh method that Mathieu mentioned yet, since I refresh the list on most command clicks in the form.

4. SmileyFtW November 23, 2018 Reply

I actually did as Mathieu suggested. Sorta. I get all of the data as a collection from a table on the worksheet using the abstractions in the example. I then convert the collection to an array; I get the subsets at the same time into subset arrays. When the filter is selected a combo box is (re)loaded with the appropriate array. The arrays are mutually exclusive in my situation so each list the combo shows is unique. When an edit of the data is "Applied" then the new model data is reimported and segregated for display in the combo. I haven't had time to consume Profex's post… I certainly will though!

5. SmileyFtW <u>December 6, 2018</u> <u>Reply</u>

If there were more than one type of "thing" that could be edited, but only one "thing" a time and the "thing" being edited is one many of those type "things". I am thinking that the ApplyChanges would accept any of the different types of "things" and decide what to do based on what is passed to it. ApplyChanges would also only want (need) to save the one edited "thing" and not the entire set of "things" the individual "thing" belongs to. Assuming the set of "things" is stored as a table (like in the example workbook) and then after the edited item is saved in that table I would think that reloading the affected table would want to be done (as opposed to managing the changes singularly in the presenter) to update the dialog.

I am thinking that the "ApplyChanges" would determine the type of "thing" passed (using TypeName) and then know how/where to save the changes assuming that each type "thing" has its own table.

Am I on the right path?

Rubberduck VBA December 7, 2018 Reply

Almost. Encapsulate the "things" into a model class, and then all ApplyChanges needs to care about is the model – which the presenter alreary holds a reference to (or it could be passed as an argument of the event) – if applying changes gets non-trivial, consider writing a dedicated, testable class for it – especially if the file system or a database gets involved.