Chapter 39: User Forms

Section 39.1: Best Practices

A `UserForm` is a class module with a designer and a **default instance**. The *designer* can be accessed by pressing Shift+F7 while viewing the *code-behind*, and the *code-behind* can be accessed by pressing F7 while viewing the *designer*.

Work with a new instance every time.

Being a *class module*, a form is therefore a *blueprint* for an *object*. Because a form can hold state and data, it's a better practice to work with a new *instance* of the class, rather than with the default/global one:

```
With New UserForm1
    .Show vbModal
    If Not .IsCancelled Then
        '...
    End If
End With
```

Instead of:

```
UserForm1.Show vbModal
If Not UserForm1.IsCancelled Then
    '...
End If
```

Working with the default instance can lead to subtle bugs when the form is closed with the red "X" button and/or when `Unload Me` is used in the code-behind.

Implement the logic elsewhere.

A form should be concerned with nothing but *presentation*: a button `Click` handler that connects to a database and runs a parameterized query based on user input, is **doing too many things**.

Instead, implement the *applicative logic* in the code that's responsible for displaying the form, or even better, in dedicated modules and procedures.

Write the code in such a way that the UserForm is only ever responsible for knowing how to display and collect data: where the data comes from, or what happens with the data afterwards, is none of its concern.

Caller shouldn't be bothered with controls.

Make a well-defined *model* for the form to work with, either in its own dedicated class module, or encapsulated within the form's code-behind itself - expose the *model* with `Property Get` procedures, and have the client code work with these: this makes the form an *abstraction* over controls and their nitty-gritty details, exposing only the relevant data to the client code.

This means code that looks like this:

```
With New UserForm1
```

```
    .Show vbModal
    If Not .IsCancelled Then
        MsgBox .Message, vbInformation
    End If
End With
```

Instead of this:

```
With New UserForm1
    .Show vbModal
    If Not .IsCancelled Then
        MsgBox .txtMessage.Text, vbInformation
    End If
End With
```

Handle the QueryClose event.

Forms typically have a `Close` button, and prompts/dialogs have `Ok` and `Cancel` buttons; the user may close the form using the form's *control box* (the red "X" button), which destroys the form instance by default (another good reason to *work with a new instance every time*).

```
With New UserForm1
    .Show vbModal
    If Not .IsCancelled Then 'if QueryClose isn't handled, this can raise a runtime error.
        '...
    End With
End With
```

The simplest way to handle the `QueryClose` event is to set the `Cancel` parameter to `True`, and then to *hide* the form instead of *closing* it:

```
Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
    Cancel = True
    Me.Hide
End Sub
```

That way the "X" button will never destroy the instance, and the caller can safely access all the public members.

Hide, don't close.

The code that creates an object should be responsible for destroying it: it's not the form's responsibility to unload and terminate itself.

Avoid using `Unload Me` in a form's code-behind. Call `Me.Hide` instead, so that the calling code can still use the object it created when the form closes.

Name things.

Use the *properties* toolwindow (`F4`) to carefully name each control on a form. The name of a control is used in the code-behind, so unless you're using a refactoring tool that can handle this, **renaming a control will break the code** - so it's much easier to do things right in the first place, than try to puzzle out exactly which of the 20 textboxes `TextBox12` stands for.

Traditionally, UserForm controls are named with Hungarian-style prefixes:

- `lblUserName` for a `Label` control that indicates a user name.
- `txtUserName` for a `TextBox` control where the user can enter a user name.
- `cboUserName` for a `ComboBox` control where the user can enter or pick a user name.
- `lstUserName` for a `ListBox` control where the user can pick a user name.
- `btnOk` or `cmdOk` for a `Button` control labelled "Ok".

The problem is that when e.g. the UI gets redesigned and a `ComboBox` changes to a `ListBox`, the name needs to change to reflect the new control type: it's better to name controls for what they represent, rather than after their control type - to *decouple* the code from the UI as much as possible.

- `UserNameLabel` for a read-only label that indicates a user name.
- `UserNameInput` for a control where the user can enter or pick a user name.
- `OkButton` for a command button labelled "Ok".

Whichever style is chosen, anything is better than leaving all controls their default names. Consistency in naming style is ideal, too.

Section 39.2: Handling QueryClose

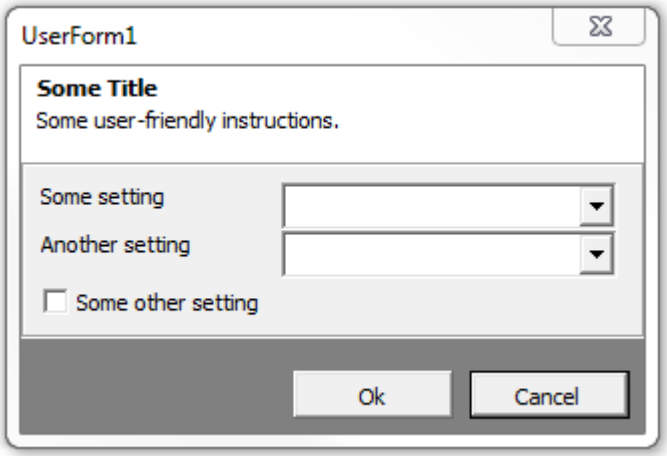
The `QueryClose` event is raised whenever a form is about to be closed, whether it's via user action or programmatically. The `CloseMode` parameter contains a `VbQueryClose` enum value that indicates how the form was closed:

Constant	Description	Value
`vbFormControlMenu`	Form is closing in response to user action	0
`vbFormCode`	Form is closing in response to an `Unload` statement	1
`vbAppWindows`	Windows session is ending	2
`vbAppTaskManager`	Windows Task Manager is closing the host application	3
`vbFormMDIForm`	Not supported in VBA	4

For better readability, it's best to use these constants instead of using their value directly.

A Cancellable UserForm

Given a form with a `Cancel` button



The form's code-behind could look like this:

```
Option Explicit
Private Type TView
    IsCancelled As Boolean
    SomeOtherSetting As Boolean
    'other properties skipped for brievety
End Type
Private this As TView

Public Property Get IsCancelled() As Boolean
    IsCancelled = this.IsCancelled
End Property

Public Property Get SomeOtherSetting() As Boolean
    SomeOtherSetting = this.SomeOtherSetting
End Property

'...more properties...

Private Sub SomeOtherSettingInput_Change()
    this.SomeOtherSetting = CBool(SomeOtherSettingInput.Value)
End Sub

Private Sub OkButton_Click()
    Me.Hide
End Sub

Private Sub CancelButton_Click()
    this.IsCancelled = True
    Me.Hide
End Sub

Private Sub UserForm_QueryClose(Cancel As Integer, CloseMode As Integer)
    If CloseMode = VbQueryClose.vbFormControlMenu Then
        Cancel = True
        this.IsCancelled = True
        Me.Hide
    End If
End Sub
```

The calling code could then display the form, and know whether it was cancelled:

```
Public Sub DoSomething()
    With New UserForm1
        .Show vbModal
        If .IsCancelled Then Exit Sub
        If .SomeOtherSetting Then
            'setting is enabled
        Else
            'setting is disabled
        End If
    End With
End Sub
```

The `IsCancelled` property returns `True` when the `Cancel` button is clicked, or when the user closes the form using the *control box*.