Awards, awards and more awards!

UK Technical Communication, Horace Hockley and Mike Austin

Communicator

The Institute of Scientific and Technical Communicators
Winter 2017

Learn about Information 4.0

Create user-orientated documentation

Do we need better authors?

Explore tools: SCHEMA ST4, SVG, Word

Creating user forms: part 1

Coding an interface for your VBA code.

By Mike Mee.

Introduction

This article will cover how to create a user form for your VBA functions. Due to the increase in complexity, the article will be spread over a few separate issues.

In the previous article (*Communicator* Summer 2017), as a simple taster, I covered how to use Word's Quick Parts to create a simple way for you to have all of your routines available at a click away.

In this article, and the follow-up ones, I will take you to the next level where you can make a VBA form appear on-screen.

The tutorial will guide you through the various steps needed to allow some interaction between the form and your VBA code, so that you (as a user) can make changes. The VBA code will then update some settings of the current document that is open.

Forms

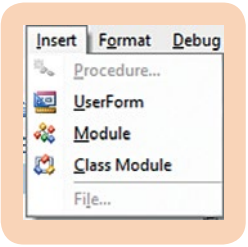
Forms are the blank grey boxes that hold the controls that form the interface to your VBA code. You need a form before you can add any GUI functionality.

You can add all manner of controls to the form, from text boxes through to drop-down boxes. But first, you need to create somewhere for all of these functions to live.

Create the form

Create a blank **macro-enabled** Word template. It should have a `.DOTM` file extension. This enables you to contain the forms, and the associated VBA code, in a single file.

Once you have your blank template document, you need to go into the VBA Editor via the Alt & F11 key combination or via the *Developer > Visual Basic* menu option. If you cannot see the Developer menu option, then you will need to switch this on via the *File > Options > Customize Ribbon* dialog.



Once you are in the VBA Editor, you need to create a blank form. Click the **UserForm** option that is under the **Insert** menu.

You will see a new form, called UserForm1, appear in the main VBA Editor window.

This blank form is where you place the controls and, behind the scenes, add the VBA code to control them and interpret their contents.

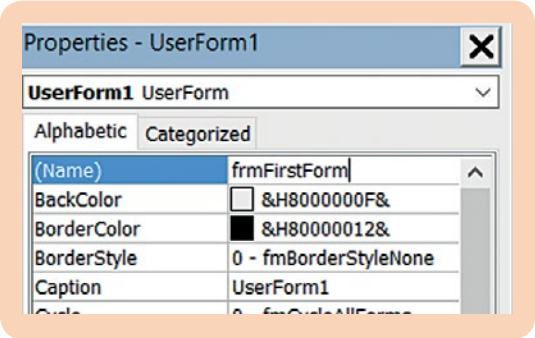
Properties of user forms and controls

There are properties available for the user forms and any of the controls that you have added to it. The majority of these will not need changing on the fly (while your code is running) and can be set as once-only configurations.

If you left-click on the form (or control) the Properties window on the left-hand side will refresh its contents accordingly.

Always rename the controls that you use on your form, as well as the form itself. It just makes your life a lot easier later on.

Click on the entry for the **(Name)** field in the Properties window and rename it to something useful, such as `frmFirstForm`. You will also need to change the title of the form that appears when displaying the form. You change this via the **Caption** parameter.



There are a lot of other changes that you can make to the form, but I will let you discover them on your own. Just remember to save your work regularly.

Initialize every form

For VBA to display your new form, it requires an `Initialize` event to be associated with every form that you use.

This event is where you will set up the default options of the controls on your form before making it appear.

Once the user closes the form, the `Initialize` event continues. It is normal practice to then interpret any changes that the users have made via the form controls.

The `<frmName>_Initialize` VBA event is not created automatically when you create a new form, so you have to insert it manually. Right-click on the new user form and click **View Code**.

By default, if you do not already have one, VBA will create the following code:

```
Private Sub UserForm_Click()

End Sub
```

Copy and paste this procedure, but change the word `Click` (after the underscore) to `Initialize`, and you are now ready to start adding the code to make the form appear.

Do not add anything to this `Initialize` routine (yet!) but add the following command to the `Click` routine.

```
Private Sub UserForm_Click()
        MsgBox "The form is alive!"
End Sub
```

When you run the VBA code, the blank form will appear and do nothing at all — unless you click the form background. Then the message box will appear.

If you click the X in the top right-hand corner of the form, it will close and you will return to the VBA Editor.

Textboxes, buttons, checkboxes, etc.

Once you have created a blank form, you can then start adding controls to it.

By dragging and dropping a control from the Toolbox (see left) onto your new user form, you will create an instance of that control that the user can see and use. It is entirely up to you how it will work.

The first item, the left-facing arrow, switches you back to the **Select** mode. This lets you choose which control on the form that you want to edit the properties of (via right-clicking the mouse) or add code to by double-clicking on it.

When adding controls to your form, try to rename the item so that it follows the Leszynski naming convention (see Figure 1).

Naming conventions

Out of habit, I define my variables along the lines of the Leszynski naming convention. This can apply to controls as well as normal variables.

Whenever I see `chkYesNo` in my code, I know that it is a *checkbox* variable and `txtName` is related to a *text box*.

http://en.wikipedia.org/wiki/Leszynski_naming_convention#The_LNC_Tags_for_VBA_Variables

Figure 1. Tips for naming controls and forms

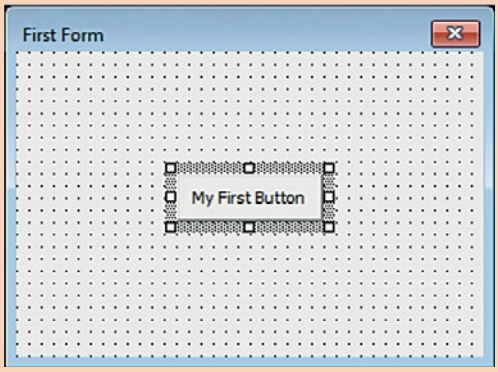
The same applies to the controls that you add to your form. The table below outlines some of the simpler controls that you can add.

I have not included all fifteen of the available controls, partly due to space but also due to some of the more advanced controls need extra background definitions and specific VBA coding, which have not been covered before. They will appear in future articles.

Control	Icon	Leszynski
Command button		cmd
Label	A	lbl
Text box	ab|	txt
Check box	☑	chk

Adding a control – command button

Drag a command button into the centre of your blank form. Rename it to something useful, such as cmdFirstButton. Next, replace the default text given to it via the `Caption` parameter, such as "My First Button" as I have done. Save the changes to the form. It should then look something like this:



Adding code behind the new control

You add code to a control the same way that you add code to a form, just double-click it.

If the control does not have any code associated with it already, VBA will create the `Private Sub<controlname_Click>` and closing `End Sub` pairing for you automatically.

By double-clicking on the new button, the following will appear in the VBA Editor.

```
Private Sub cmdFirstButton_Click()

End Sub
```

Note: When you double-click on a control, if it is new, or an existing one that does not have any code associated with it, the new code routine is inserted at the very top of the form's code module. When your VBA project is growing and growing, this can be a bit of a pain.

The only solution that I have found is to cut the empty procedure that is created and paste it wherever you want it to appear in the code module. The next time you double-click on the control, VBA remembers where you placed it in the module and jumps to it.

Whatever code you insert in here will only be run when the button is clicked. So for the first instance, the code will not be that exciting. A

simple message box will suffice for now to show that the button is working.

Alter the code to the following:

```
Private Sub cmdFirstButton_Click()
        MsgBox "You clicked my button!"
End Sub
```

So far, this program is very basic but if you close the code module and run the form again, you will see that nothing will happen **until** you click on the new button.

First controls: labels and text boxes

Label and text controls are the most basic items that you can add to your form. Their uses are slightly different, as is the VBA code to control them. They can be used as items to display content, or be made interactive. The choice is yours as it will be your form and interface that the user will see.

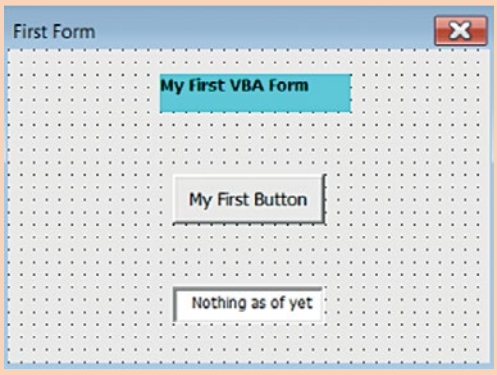
Within my Word Toolbox, I use the labels mostly for the field titles on a form, and text boxes are used to hold data that changes. A few of the text boxes are clickable or will call up a function if their contents change, but these are a bit above the baby-steps usage that are in this first article.

Adding a control: labels and text boxes

In the example screenshot below, I have added a label above the button and named it `lblHeader`. This will be the title text that will shout out to the world what the name of my form is.

The content of the label is altered via the Caption parameter. I have also changed the background colour to a light blue and made the font bold via the `BackColor` and `Font` parameters accordingly. It is then up to you what kind of layout tweaks you want to make.

Underneath, I've added a text box and named it `txtInfo`. The content of this text box will change via our code, but initially it contains the text 'Nothing as of yet'. This is set via the `Value` parameter.



The above image is how my example looks. Yours will vary depending on what extra tweaks you might have decided to use.

Note: Labels store their text content via the Caption parameter, but text boxes use the Value parameter. Many times in the past I have typed in the wrong parameter name, only to see my program crash and burn on the next run!

Adding code to the label and text controls

Once you have added both the label and text boxes to your form, they will sit there doing absolutely nothing, until you add some code to interact with them.

The quick code change listed below will demonstrate how VBA events work in relation to the controls that you add to your new forms. Within certain limits per control, you can add various types of interactivity to your form. These events are used a lot in my Word Toolbox.

An easy example will be to set up the text box to interact with the label. Whatever you type into the text box, will appear in the label at the top.

To add the code for this, you double-click on the new text box and, by default, the code window will display this:

```
Private SubtxtInfo_Change()

End Sub
```

The default code creation is handy because we want to add code to the text button so that when the contents of it change, it will be handled by this routine. Add this single line of code in between the `Private` and `End Sub` commands, and then run the form.

```
lblHeader.Caption = txtInfo.Text
```

Click on the text box and start typing in something. As soon as you touch a key on the keyboard, the contents of the text box are also transferred to the label simultaneously. This shows the event-driven side of VBA, albeit a simple example.

Next article

I was hoping to be able to add the use of the check boxes and option/radio buttons, but I have run out of space. These controls will be covered in the next article, along with the use of frames to help tidy up your VBA form as it grows in size. C

References and further reading

Mee M (2015) 'Variables and screen output/input' *Communicator*, Autumn 2015: 44-47

Mee M (2017) 'Using the Quick Access Toolbar' *Communicator*, Summer 2017: 52-53

Microsoft 'Declaring Variables' https://msdn.microsoft.com/en-us/library/office/gg264241.aspx?f=255&MSPPError=-2147217396 (accessed April 2016)

Mike Mee MISTC
Contract technical author
E: mugukmail@gmail.com
T: @Mug_UK
W: www.mikestoolbox.co.uk
– my toolkit for Word 2007-2017