Plain language – what is it?

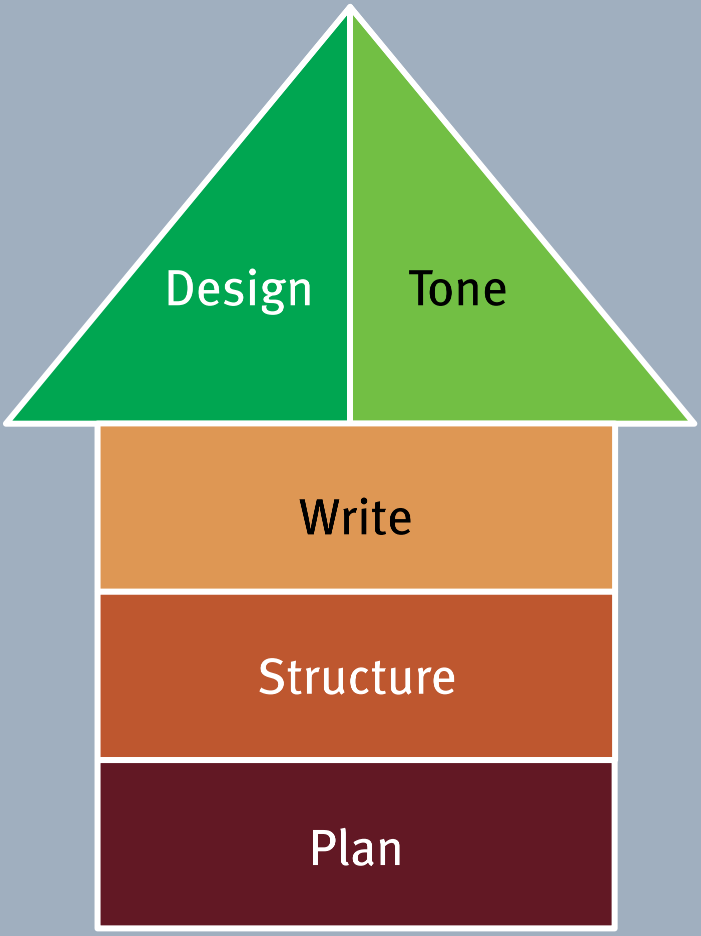
How do you apply plain language to your documentation?

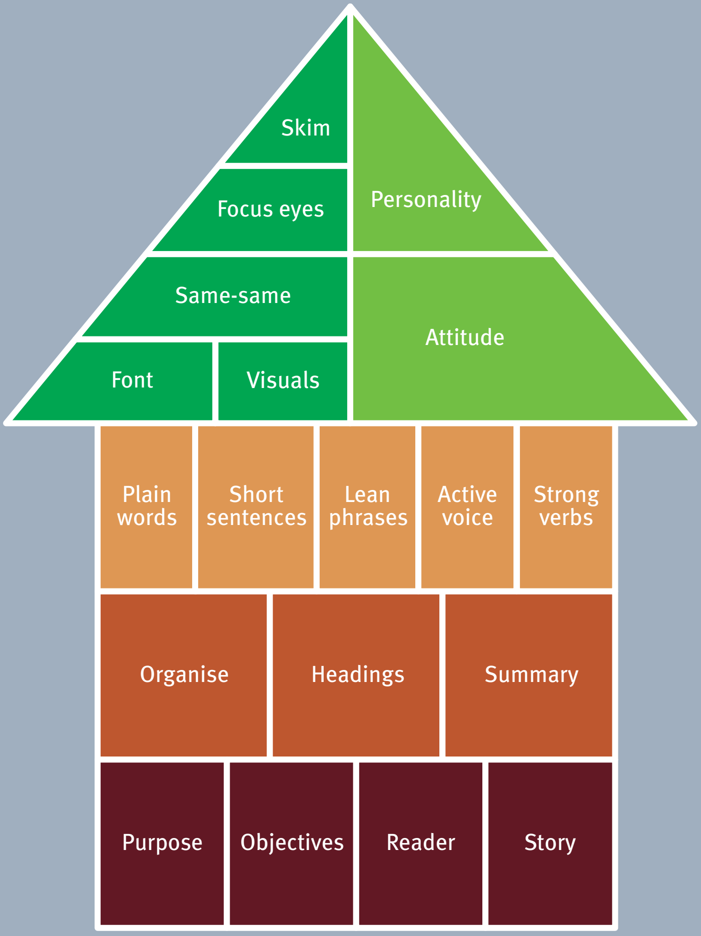
Communicator

The Institute of Scientific and Technical Communicators
Spring 2018

Guidelines for e-learning localisation and; video

Learn more about Gestalt principles for designing visuals





Simplified - an approach to plain language as a process

Read a case study on Miele and SCHEMA ST4

Re-ignite your career with MadCap Flare

Creating user forms: part 2

Coding an interface for your VBA code.
By Mike Mee.

Introduction

In the second part of this series of articles, you will learn about more control types that you can add to your own user forms.

I will be covering frames firstly, as these help to straighten out, or neaten, your form layout designs.

After that I will cover check boxes and option buttons, also known as radio buttons. These controls are frequently used on VBA-based user forms and these last two items come in handy when dealing with options or flags that you want to present to the user.

Naming convention

As with the previous article, each type of control has a three-letter prefix that you can use to help you remember which control each variable is referring to when you interpret the actions.

Table 1. Controls, icons and naming conventions

Control	Icon	Leszynski
Frames	▭	fra
Checkbox	☑	chk
Option button	◉	opt

Adding new controls

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

As shown in the previous article, by dragging a control from the Toolbox and dropping it onto your new user form, you will create an instance of that control that the user can see and use.

Frames – keeping your form neat

Generally speaking, frames can be used to keep related controls together, making the form's appearance neater.

Instead of placing all of your controls onto a form and just hoping your end user will work out what to do with it, you can point them in the right direction. This makes your program flow easily too.

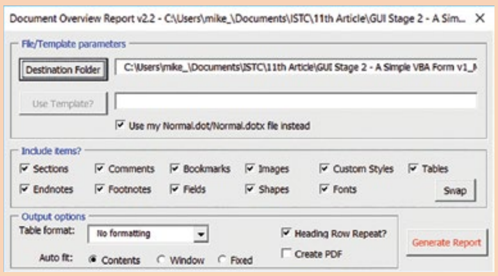
Frames can also ensure that option buttons are mutually exclusive within a specific frame. You will not be able to select more than one option button at a time; I cover this later on in the article.

Frames – a working example

In the following example, Figure 1, you can see how multiple frames are used to separate areas of the user form from each other.

Each frame includes a title (displayed in blue) so the end user can tell which part of the form they are looking for. This also helps me when I am writing up the functionality in the manual, as you can refer to each frame's contents to direct the user.

There is a new function in the next full release (v2.2) of my Word Toolbox. It allows the user to configure what information is extracted from the current document and subsequently appears in the Word/PDF report that is generated.



Document Overview Report v2.2 - C:\Users\mike_\Documents\ISTC\11th Article\GUI Stage 2 - A Sim...

File/Template parameters
Destination Folder: C:\Users\mike_\Documents\ISTC\11th Article\GUI Stage 2 - A Simple VBA Form v1_
Use Template?
☑ Use my Normal.dot/Normal.dotx file instead

Include items?
☑ Sections ☑ Comments ☑ Bookmarks ☑ Images ☑ Custom Styles ☑ Tables
☑ Endnotes ☑ Footnotes ☑ Fields ☑ Shapes ☑ Fonts

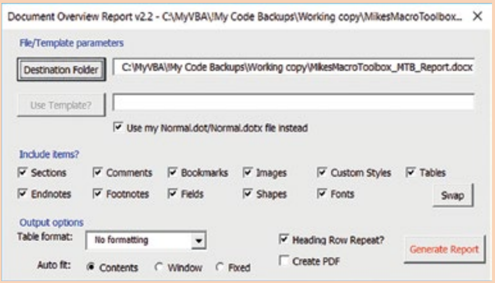
Output options
Table format: No formatting
Auto fit: ◉ Contents ○ Window ○ Fixed
☑ Heading Row Repeat?
☐ Create PDF

Figure 1. Form example with frames
Source: Mike Mee's Word Toolbox v2.2

You can see that there are three separate frames on the form: **File/Template parameters, Include items?** and **Output options**.

The form layout appears neat and tidy thanks to the use of frames which allow you to group related controls into separate areas.

The same form is included in Figure 2, but all of the lines from the frames have been removed.



Document Overview Report v2.2 - C:\MyVBA\\My Code Backups\Working copy\MikesMacroToolbox...

File/Template parameters
Destination Folder: C:\MyVBA\\My Code Backups\Working copy\MikesMacroToolbox_MTB_Report.docx
Use Template?
☑ Use my Normal.dot/Normal.dotx file instead

Include items?
☑ Sections ☑ Comments ☑ Bookmarks ☑ Images ☑ Custom Styles ☑ Tables
☑ Endnotes ☑ Footnotes ☑ Fields ☑ Shapes ☑ Fonts

Output options
Table format: No formatting
Auto fit: ◉ Contents ○ Window ○ Fixed
☑ Heading Row Repeat?
☐ Create PDF

Figure 2. Form example without frames
Source: Mike Mee's Word Toolbox v2.2

As you can see, the form in Figure 2 does not look anywhere near as appealing without the frames. Also, due to the lack of frames on the form, if you wanted to disable a group of related controls, you would have to individually disable every control via VBA. Whereas, if the controls were in a frame, you could just disable the frame itself.

Frames – adding them to a form
Using the form you created using the previous article (*Communicator*, Winter 2017), see Figure 3, expand the form's usable area and add a check box to the form and name it `chkContent`. Change the check box's caption to 'Content'.

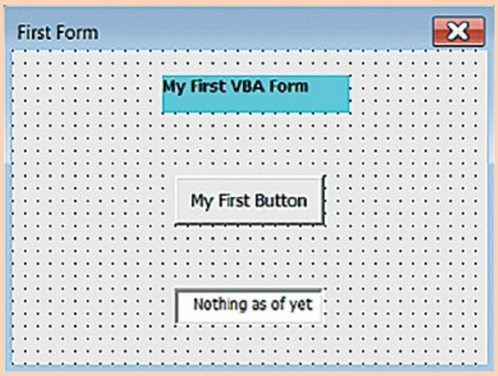


Figure 3. Form created in Word VBA macro tips article: *Communicator*: Winter 2017

Next, create a new frame. If your form's design has anything else that would look neater inside your new frame, then relocate it accordingly. For example the check box and the text box look better inside the same frame as shown in my example - Figure 4.

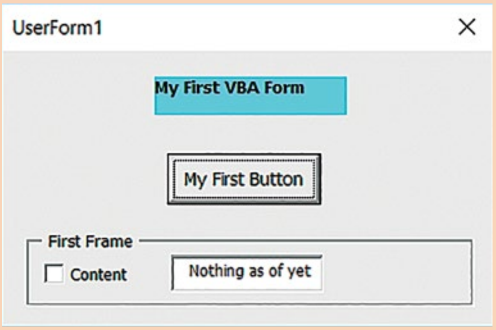


Figure 4. Form created in Word VBA macro tips article: *Communicator*: Winter 2017

I have put the frame under the button and then moved the check box and text box inside the frame. My frame is called `fraFirstFrame` and the caption is 'First Frame'.

Frames – turning them on and off
Frames, like all other controls, can have VBA code associated with them. You can disable them, which makes all controls within the frame unavailable to the user.

This saves you from having to include additional code to enable or disable (as appropriate) individual controls within a user form.

However, if the frame is enabled, you can still disable a single control; you can do this without disabling the entire frame.

The code to disable a whole frame is very simple: `fraFirstFrame.Enabled = False`. This will grey out the whole frame and everything within it. **None** of the controls can be accessed whilst the frame is disabled.

Changing this parameter to `True` will reenable the frame, allowing the user to interact with whatever controls are inside it.

Tip: *If you ensure that the start-up code (via the* `Initialize` *procedure – see below) associated with the form always enables all of the frames before the form is shown, it will not show what the form looked like after the user has finished with it previously.*

```
Private Sub UserForm_Initialize()
        fraFirstFrame.Enabled = True
End Sub
```

Believe me, this happens a lot when you are first starting out with forms. The same also goes for any other controls too.

Frames – changing the frame's title
Another function for frames is the `Caption` parameter. You can alter the text of the frame's title using this parameter – I use it a lot in the Word Toolbox when displaying a count of a particular item discovered in a document. If none of the relevant items are found, it displays a zero in the frame counter and then disables the frame. This prevents the end user from trying to delete non-existent items, such as a table or image.

Frames – removing them (a warning!)
This section of the article is the one that should have a big red box around it as a warning for when you need to remove a frame from a form.

Every control within the frame will be deleted at the same time as the frame! However, any associated VBA code will be left behind.

If you want to keep any controls that are currently within the form, resize the form so that there is some extra space. Move the controls out of the frame and into the extra space. Then delete the frame and move the controls back to where you wanted them.

It is tedious having to do it this way, but not as tedious as having to recreate the controls you have just deleted. Especially when the Undo option (or Ctrl & Z) does not always work – as I've found out a few times in the past with my Word Toolbox development.

This covers the basics of frames. The next sections deal with adding and interacting with check boxes and option buttons.

Check boxes
Check boxes are the easiest way of showing the user if a value is true or false. It can also be

used to allow the user to switch an option on or off. Similarly, it can also be used to detect if a particular setting in the document, or Word itself, is on/off and the check box display is altered accordingly.

Adding a control – check boxes

If you have already added the check box that I mentioned earlier on the form, then do not add it again. For everyone else, add the check box to the frame and name it `chkContent` with a caption of 'Content'.

Double-click on the new check box to display the VBA code window. The following code will have been automatically created for you.

```
Private Sub chkContent_Click()
End Sub
```

Note: By default, this will always be a 'Click Event' which only happens if the control is clicked on.

We are going to add a simple on/off function to the code that will enable/disable the text box depending on the current status of the check box.

Note: Check boxes are selected or cleared, they are **not** ticked/unticked! Check your Microsoft Manual of Style if you do not believe me. I have spent years fixing that kind of mistake in many, many manuals and helptext gone by.

Adding a control – check boxes

First we need to alter the `Private_Sub chkContent` type from a `_Click` to a `_Change` as we want the code to react when the user changes the status of the check box. The first line of the subroutine should now look like this one: `Private Sub chkContent_Change()`

The next stage is to add in the code that will toggle the enabled status of the text box depending on what the current status of the check box is. This is done using this snippet of code.

```
Private Sub chkContent_Change()
If chkContent = True Then
    txtInfo.Enabled = True
Else
    txtInfo.Enabled = False
End If
End Sub
```

If you are a regular reader of this column, you might remember an article I did on optimising VBA code (*Communicator*, Spring 2017). The five lines of code that enable or disable the text box can be replaced by a single line of code like this:

```
txtInfo.Enabled = IIf(chkContent = True,
True, False)
```

Whichever of the two code snippets you implement, when you run the code you will see that, depending on what the original status of the check box was, that sometimes the initial toggle of the text box is not recognised. This is easily fixed by adding an additional line of code to the UserForm_Initialize subroutine which forces the text box to be disabled every time the form is opened. You can place it before or after the code discussed in the previous section covering frames.

```
Private Sub UserForm_Initialize()
        fraFirstFrame.Enabled = True
        txtInfo.Enabled = False
End Sub
```

Now if you run the code, the form will default to showing the text box in its disabled state and clicking the check box once will enable it. Clicking it again will disable it.

That is your first check box running on a VBA form. It is up to you how many more you add to this, or your own forms. Just remember that if it needs to be reset to a default value before the form is displayed, place that code into the `UserForm_Initialize` subroutine.

Option buttons

Option buttons are similar to check boxes in that they allow a user to make a choice, but they are not just limited to True or False. This is because you can have more than two option buttons within a form (ideally inside a frame) providing the user with a choice of options.

However, please note, that option buttons, when included within a single frame, are mutually exclusive! You cannot select more than one option button at a time. This also limits your form design as each set of option buttons needs to be included within a separate frame on your form.

Adding a control – option buttons

Using the form you have created so far, expand the form's depth so that you can include another frame underneath the existing one with the check box.

In this new frame, add three new option buttons, called `optRed, optGreen` and `optBlue`. Their labels are Red, Green and Blue accordingly.

This is what my version of the form looks like now – I have not added any code to the option buttons as of yet. But if you run the form, you can see how the mutual exclusivity of each option button works.

As per the check box example earlier on in this article, if you double-click on each of the option buttons, you will create (on the first attempt) a new but empty procedure in your VBA code. The VBA code window will look like this for the option buttons.

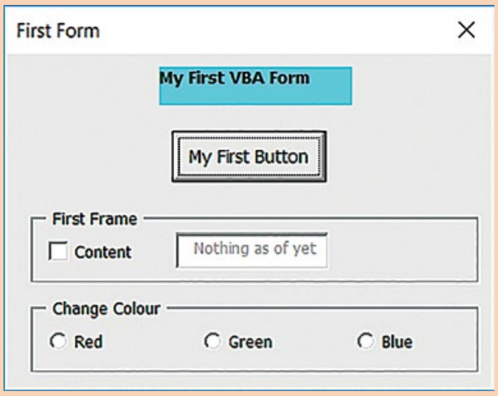


Figure 5. Form with option buttons inside a frame

```
Private Sub optRed_Click()
End Sub

Private Sub optGreen_Click()
End Sub

Private Sub optBlue_Click()
End Sub
```

We are going to change the background colour of the text box in the frame above, depending on which option button was clicked. The parameter that controls the text box's background colour is `BackColor`.

The code to insert into all three of the option buttons is:

```
txtInfo.BackColor = <colour> (in Hex)
```

Using hex enables you to select any colour you choose. Alternatively, for the colours Black, Blue, Cyan, Green, Magenta, Red, White and Yellow you can use a colour constant, see Table 2.

Table 2. Colour constants

Colour	Constant	Hex Value
Black	vbBlack	&H0
Blue	vbBlue	&HFF0000
Cyan	vbCyan	&HFFFF00
Green	vbGreen	&HFF00
Magenta	vbMagenta	&HFF00FF
Red	vbRed	&HFF
White	vbWhite	&HFFFFFF
Yellow	vbYellow	&HFFFF

These constants can be used anywhere in your code in place of the actual values.

You can use the colour selector option next to the `BackColor` parameter to find any colour that you want to use, then copy and paste the hex values into the VBA code.

If you are not feeling that adventurous just yet, here's the code for all three option buttons using hex:

```
Private Sub optRed_Click()
    txtInfo.BackColor = &HFF
End Sub

Private Sub optGreen_Click()
    txtInfo.BackColor = &HFF00
End Sub

Private Sub optBlue_Click()
    txtInfo.BackColor = &HFF0000
End Sub
```

Here's the code for all three option buttons using colour constants:

```
Private Sub optRed_Click()
    txtInfo.BackColor = vbRed
End Sub

Private Sub optGreen_Click()
    txtInfo.BackColor = vbGreen
End Sub

Private Sub optBlue_Click()
    txtInfo.BackColor = vbBlue
End Sub
```

When you run the form now, you can click on each option button and the background colour of the text box will change accordingly. However, there is no code present to reset the background back to the original white colour once the form is running. I will leave you with the challenge to include a fourth option button to do that.

Next article

The third part of this article will take a pause from describing the controls and will instead take apart a form from my Word Toolbox.

This will show you, warts and all, a real world example of a form and the VBA behind it. It will be the About box from my Word Toolbox.

This form contains a few different controls, including clickable images (which have not been covered yet), along with how VBA controls/interacts with the form.

That is, assuming, that I am not spending too much time with my new ZX Spectrum Next (a modern remake of the classic 1982 home computer), which should have arrived by the time this article appears in print. C

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

References and further reading

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

Mee M (2017) 'Optimising your VBA code' *Communicator*, Spring 2017: 37-39

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 January 2018)