overview of user forms and input

Some sections to include here:

- using the Application.InputBox to get different types of input (especially Ranges)
- creating a UserForm and wiring up events
- interacting with the spreadsheet from a UserForm
- a quick summary of the different pieces, when they're useful and specific/common properties

at the end of the day, there is not much too special about forms other than knowing what the different pieces are

this should be a relatively quick chapter since most of the reference material is elsewhere

introduction to UserForms

This chapter will focus on how to use UserForms to create interface that allow the user to interact with your VBA code. UserForms can be used from anything to simple text inputs to very complicated forms. There is really no limit to what you can do with UserForms, but at some point you will hit the limit of what you want to do inside the VBE. Some folks will push the limit and develop fully featured programs in Excel. I'd highly recommend you not do that and instead use UserForms to augment a good usage of VBA with

useful interactivity.

When considering whether or not to use UserForms, there are a handful of pros and cons to using them.

Pros to using UserForms

- Provide a form to display/edit user input independent of Excel
- Provide for much better interaction with the user via “normal” programming events click, keyboard input, etc.
- Allow your program to collect several pieces of information before completing an action, especially if some real time process of the information is useful
- Provide a form that can “sit” on top of Excel and provide helper functionality or application specific functionality

Cons to using UserForms

There are a number of alternatives to creating a UserForm that are worth considering before committing to UserForms for a specific application. Those alternatives and the cons against UserForms include:

- Editing the code of a UserForm can be a bit of a nuisance because you have to flip between design and code views
- The InputBox provides a simple way to collect a number of different input types without needing to

create your own form. For simple inputs (including Ranges), the InputBox is typically simpler and more consistent across applications

- Some types of inputs (including lists) can be easier to manage in a non-VBA context by using default Excel features. For example, you do not need a ListBox on a UserForm if you can put a Table somewhere in the spreadsheet.
- If you are trying to use some form of version control on your source code, UserForms are very difficult to manage and version.
- If you want to provide buttons to perform an action, the Ribbon can be much more robust. Of course editing the Ribbon can be a different pain, and I've gone the other way on this point before.

Once you've decided you want to use a UserForm to provide for user input/interaction, there are a number of areas that are important. Those areas from start to finish are:

- Adding a UserForm to an existing Workbook or addin
- Using a Sub to make a UserForm show up using a keyboard shortcut or button or some other means
- Adding controls to UserForm and using those inputs to drive VBA
- Working with a UserForm to process input and update the UserForm
- Using the non-control events to provide interactivity

creating a UserForm

Creating a UserForm is a simple process: open the VBE and then right click to Insert -> UserForm. This will give you a default UserForm that is blank, has a default name, and is a default size. If you're lucky, this will also open the UserForm for editing and show you the toolbox which provides controls to edit. Once the form is created, there are a couple of things which you should do immediately, before you forget:

- Change the name of the form to something more useful than UserForm1
- Change the `Caption` on the form to something better than UserForm
- Consider changing the `ShowModal` property if you know you do not want a modal dialog

Once those items are done (or decided against) you can start adding controls to the USerForm and changing its size.

TODO: add pictures of the steps

making that USerForm show up

Once your UserForm is created, there are a couple of ways of showing it on screen:

- Run any code from the VBE that is contained within the form. This will show the form.
- Create an instance of the form somewhere and show it

For those two methods, the latter is really the only one that will work for user applications or other “real” uses. If you are simply testing or doing things for yourself, then hitting F5 in the VBE may not be a large ask.

For the former, see the code below for an example of how to show the form.

```
DIm frm as UserForm
Set frm = New UserForm

frm.Show
```

adding controls to a UserForm and wiring them up

Once your UserForm is created and the defaults are changed, your next task is to do something useful with the form. To that end, you will quickly need to add controls to the form and then wire those forms to useful actions. Adding the controls to the form is a straight forward process: show the Toolbox and then drag the items onto the form. Once you have dragged out a button, text box, and possibly a ListBox, you can simply copy and paste the previous items and avoid the Toolbox all around. There are a handful of controls that are not in the Toolbox by default. My strong advice here is to not use those controls if you are deploying this addin. Inevitably, some user will not have the OCX file or whatever is required to make it work. Just pass. Having said that, you may need to add a Date picker or possibly a RefEdit which are not included

here (TODO: is that true about RefEdit?).

Dragging a control onto the form is fairly easy compared to the actual task of making a control do something useful. Below is a quick primer on how the different controls work, which properties are important, and will give you a guide for accomplishing 90% of what can be done with forms.

CommandButton

The CommnadButton or simply button is one of the most common controls to use. Its use is simple, known to everyone, and easy enough to program against. A button does one thing: get clicked. The event you want to know about is the `_Clicked` event. Fortunately, the VBE will automatic create and wire up this event if you double click the button on the Designer version of the form. This makes it dead simple to create the button code that you want: just double click the button.

Note that the default event will be created with the current name of the control. To avoid this, you need to change the name of the button before you create the event. Be aware that VBA and the VBE are not that smart with respect to naming things nad wiring up changes. If you change the name of the button after you create the event, your event will not work. You should not change the button name after creating the vent (or plan to recreate it).

Other properties of the button that might be used:

- Value - will change the text that is displayed (TODO: is this right?)
- Enabled - will change whether the button can be pressed and will change the visuals. Useful when you want to show that an option could be possible but is not currently allowed or enabled
- Formatting and other visuals - you may change this on the property editor but it is far less common to modify the formatting once you are running. It can be done but is not common.

That's it; buttons are simple.

CheckBox and Radio

The CheckBox and Radio are cousins (or siblings?) of each other and will be dealt with at once. They allow for a Boolean selection of an option. For the Checkbox, you are allowed to indicate the on/off state of a given button. For a Radio, you are allowed to indicate the on/off state for a single option *within a group of options*. The main thing to note about the Radio is that by selecting one item, you will deselect the others. In this way, the uses of these two controls maps naturally to the tasks you are likely to see.

Aside from the Name, the main items to deal with are:

- Clicked event - just double click to get this one

- Value - note you get this by default using the name, but it will include a Boolean of the selected state
- Enabled - can be used to disable the control

That's about it. You can change the formatting and other stuff, but these items typically exist to get an input and get to the real work. They are very common when you are providing options to the user or otherwise want to direct downstream If/Switch statements.

Beware that the Click event may be changed multiple times depending on how it was triggered (TODO: is that right?).

TextBox

The TextBox is another simple one: it provides a means for the user to provide some text input. They work great for a range of things including input and output, although input is more typical. The idea is simple, the user provides a string and you use it somewhere. The properties to know:

- Value - this gets or sets the value that is displayed
- Enabled - can be used to disable the control (TODO: same as readonly?)

In terms of events, the main one to watch for is the KeyPress (TODO: or changed?). The idea is simple, if you want to track the input of the user, you tag along for that event and can respond to their key presses.

The common uses of this are:

- Close a form or clear an input when ESC is pressed
- Do some action when ENTER is pressed
- Provide some form of validation or checking as the user types to either modify their input (e.g. ignore dashes) or otherwise update the UI based on their input.

TODO: add some addl onctent here about the event and its callback/parameters

That's it.

ListBox

The ListBox is one control that has a number of options and a means of using it that are less obvious than the other controls. It's a shame really that the ListBox is so unintuitive in VBA because it is quite powerful and other programming languages have handled this better. THe idea behind a ListBox is that it provides a list of items whose use can vary according to what you want. Some common applications include:

- Allow the user to select from one or multiple options in a list
- Provide some output to the user (and possibly then use that output as the input for next step)

THe input/output decision here is somewhat critical because the things that will annoy you about the

ListBox break on this point. If you are collecting input, then really you have to also deal with output because at the end of the day, you have to put something in the ListBox in order for a user to select it. Once you've handled the output stuff, then determining which items have been selected by the user is straightforward enough. Therefore, covering the output part is a good starting point.

To put items into the ListBox, you need to modify the List collection on the object. There are two ways to do this:

- Directly, via the List object
- Indirectly, using the `AddItem` command

Either way you go, you have a couple of decisions after adding the item: what text do you want displayed for the item and do you want multiple columns? If you are dealing with a single column, then you can simply add the text in the call for an addition and that's all. IF you are working with columns, then you will need to do two things:

- Set up the columns (using the editor or via commands) (TODO: add pictures or code here)
- Call the command to set the fields using the row and column number (TODO: add some code)

ALthough I have described a simple process here, oftentimes, you will deal with something that is more complicated. THe issue comes when you want to maintain some reference to an object but you are required

to use a string for display purposes. This means that you need some means of maintaining that reference back to the object. There are options for dealing with this:

- Rely on the index of the objects matching (and not changing) and simply use the row index
- Create a Dictionary that stores the link between the string and the object
- Use some other object or Collection that can reference the object back to the string
- Serialize the object into the ListBox value (if multiple fields, join with a | or similar)

Each of those approaches has its pros and cons, btu the main idea is that you are often forced to deal with something that is typically much easier in other languages. My general approach is to rely on row index if I know that changes are not possible. This is common for a lot of code since yout ar likely to control both side. If that is not ideal, then you can typically find some way to store a reference between the display value and the object using a Dictionary.

Once you hav eth information in the ListBox, you can simply iterate the `Items` by index nad check the `Selected(index)` property to see if the item is selected. Note that if you do not allow multiple selection, then you can also use the `SelectediNdex` property (TODO: is that right?).

TODO: add some code here to demonstrate iterate through a ListBox

Although this section has the most text, the ListBox is not always a pain to deal with. Typically they are much

better than the alternatives (like using the Excel spreadsheet somehow) but require that you remember some boilerplate for accessing and changing items.

Other Controls

There are a couple of other controls that you may see that are summarized here:

- Label: these don't do much other than provide some fixed text when could be changed later (I rarely ever do it)
- RefEdit: this control technically allows you to select a Range from Excel. They are quite buggy. Depending on you main goal, you may of much better to use `Application.InputBox(Type:=8)` to access a Range.
- Tabs: these can be helpful for organizing a complicated workflow. You will find yourself wanting to change the active tab and possibly limit access to later tabs.
- Wells?, whatever it's called, there allow you to Group controls. These may be required for a Radio to work like you want (if you have multiple sets of Radios on a single form).

doing actions on a UserForm

Section will focus on performing commands on a UserForm without leaving the UserForm. This looks mostly like normal forms programming.

Programming on a UserForm is one area of VBA that is largely independent of Excel. Yes, you are able to access the Object Model from a form, but most of the programming on a form is simple related to the form. This is also the one area of VBA where if you have done it before in another language, your experience will transfer nearly 1:1. Forms programming is largely all about building a good UI that meets your needs. You can do a lot with VBA in terms of events and other sorts of dynamic programming, but most of the itme you just make the form and get on with the real work.

Some of the comment things you will want to of when programming on a form include:

- Creating event handlers
- Parsing and responding to user input
- Populating data in the UserForm from the Object Model
- Accessing Properties of controls to change or use

Event Handlers

Event Handlers are at the core of User Forms and making them useful. To be clear, your Form will do nothing without events. You could it to display static content from the designer mode, but it will do nothing useful. To make your Form become useful, you add Controls to it and then add Events to those Controls. Event Handlers are the glue (or wires) that take the actions performed on Controls and direct them somewhere useful. Events control everything from Clicking, Loading, Typing and everything else. Each Control has a unique set of events depending on what it can do, but in general, there's a bit of overlap between different controls.

To add an event handler, there are a couple of options:

- Double click on the Control in Design Mode, and you will get the default event handler created
- Go to the code view, and select the Control and then Event you want from the drop downs (TODO: add image)
- Type the Event handler based on the named of the Control and the event you want

If you know the default events, then option 1 is as good as the theories. IF you want to see a list of events before creating one, then you will go with option 2. You will pretty much never type the event handler out by handler unless you are copying it from somewhere else.

Once you have created the handler, you simply add the code that you want to fire in the event. One good tip here is to use the event handler to call other Subs. It's a good habit to not put logic or other execution based code into Even thNadlers. The reason for this is that you may want to perform the same action from multiple events. Putting the code in a handler makes it difficult to reuse the code because some ehadnlers have parameters nad other details that make it hard to arbitrary call them. Of course, I regularly put code into event Handlers, but at least I know I should avoid it. I am constantly reminded of why to avoid it when I have to extract code from one event to put into a Sub to call from another event.

One important note about Event handlers is that the handler can have some number of parameters that are included in the handler signature. These parameters ar etpyically used to pass along information related to the event. For example the key press event contains the key code of the key that was pressed. The Click event however has no parameters. The presence of parameters is easy to check when the VBE creates the handler for an event since it will give the parameters.

TODO: given an example of using Handlers?

TODO: include a blurb about the Initialize event (if it was not addressed earlier)

Processing User Input

User Input on a User Form is one of the most critical aspects ofm aking them. IT is less common to use a Form purely for output of information (although that is done). typically, you use a Form to provide input in a format that is easier to use than the default Excel interface. There are a ahdnful of Controls which are viewed as collectors of user input. You can then process than input in an Even tHandler or in other code which accesses the properties of the Control. Those common controls are:

- TextBox: Works great when you want to control a single value from the user. You can then parse the string into a number or whatever else you need
- ListBox.: Works great for allowing the user to select form a list from still being able to see multiple items in the list. Also supports multiple selection
- COmboBox: Same as ListBox but the control collapses to a single line when you are not selecting items. Does not allow for multiple selection.
- CheckBox or RadioButton: Allow the user to make a selection between choices while seeing the choices
- Button: Really allows a user to input a single click
- RefEdit: Not recommended but it allows you to select a Range from the Spreadsheet.

- TODO: any others (number bumper?)

For each of those Controls, you have a number of events which can be used to process the input as it comes in, or you can process the Properties of the Control once other code is running. One common pattern is to allow the user to input data into a number of TextBoxes, hit a button to run some action, and ten process all of hta input in one step after the button press. Another way to do the same thing would be to process and validate the input as it comes in, providing an error message if bad data was input.

For most of the Controls given above, you will find a `Value` property which gives either the Text of the Control or the selected state. The one exception to this is the ListBox which requires a little more work to get the Selection. For the ListBox, you need to iterate the items and check if the `Selected(index)` property of the ListBox is `True`.

TODO: add an example of using Value

Once you have the user input, it will typically be a `String` or a `Boolean`. To do something with these inputs, you will need to parse them into the desired types if not a string. The most common transformation is to parse a number from the string. This is done with `CInt` or `CDbl` which will *C*onvert a String into a Integer or Double. You will get an error if the string was not parseable. If you do not need a number, there are a couple of other “C” functions:

- CBool
- CDate
- CErr
- TODO: add others, and descriptions

Accessing the Excel Object Model

From a UserForm, you have full access to the Excel Object Model. This can be very handy if you are trying to access information from the USerForm to determine what information to show in the Form. It can also be helpful if you want to make changes to the underlying spreadsheet from a USerForm without leaving the form. Both of those options are very common and very easy to do with UserForms. In general, any code that can run without a USerForm present can be run with a USerForm. There are some limitations when it comes to the user's ability to Select items with a From visible, but you are not limited in calling the same commands from VBA (TODO: is that right?). The exception ehre is that if the form is `ShowModal = False` then the user is able to make selections while the from is visible.

There is no real limit to what you can do from a SuerForm. A couple of examples to give you a feel:

- present a list of all open Workbooks so that they user can select which one that want to process
- Create a form that can process all of the selected Charts.

- Present a ListBox with the unique values from all of the AutoFilters that are active. Allow the user to selectively remove or change those filters without having to use the normal drop downs.

Accessing control Properties

THe final piece of Forms programming is somewhat meta: allow the UserFOrm code to change the USerFOrm. There are a couple of obvious reasons you might want to do this:

- Change the position of the USerFOrm (center on start)
- Enable or disable a button or other control based owns ome input. You can extend this to making things visible or not as well.
- Change the text, format, or other visual detail of a Control based on some other state or user input.

TODO: add the code for centering a UserForm.

IN addition tot hose simple conners, you also have the ability to danmically create controls on demand. This makes it possible to add/remove controls to the USerForm as needed. This can be helpful if you want to create Control based on some proeprt yof the Worksheet but where you may not know how many times to do it in advance. For example. maybe you want to provide a LIstBOx with unique values for each column that was selected. IN advance, you may not know the column count so you need to create ListBoxes on

demand. This can be done with UserForm programming.

TODO: example of create a Control from scratch