overview of adv processing

Advanced processing should include some recipe type sections that go through the more advanced aspects of working up VBA code.

This could focus on:

- Speed improvements and how to do it (disable screen, events, calculation) and how to undo it
- Working with arrays of values instead of outputting a cell at a time
- Cranking through an entire automated workflow without user interaction: creating new workbooks, worksheets, charts, formulas and then outputting it all to PDF
- Focus on the interplay of manual steps and code (sometimes you have to run part of the code to see what to do next; other times you can sit down and type the whole thing out)
- Cleaning up macro recorder code (some discussion about what works well/doesn't)
- How to avoid `Select` and why
- Using `DoEvents` to wait a set amount of time
- Using `Application.OnWait` (?) to do some thing at a regular time
- Parsing through existing formulas or values and manipulating with confidence
- Reading and writing to external files

- Working with the file system to do some processing
- Running through a folder or batch of files and doing something with each one
- Structuring code in a way that the different pieces can be called on their own
- Going through a workflow that involves using other office products
- Strategy for identifying cells using Styles and working through them; effectively a tag feature

The long list of sections here says that maybe there is enough code to put together a couple of "case study" type things that break down the development of an entire workflow. This could related to charting/processing or some other thing.

TODO: consider going through StackOverflow answers to see what the most common slightly advanced topics are that come up

some thoughts on creating a workflow

If you are sitting down to create an advanced workflow, there are a handful of things to consider. The list that follows is not complete nor is it meant to include items that are always relevant. The problem with these lists is that with a general programming environment like Excel, it's impossible to describe everything to consider. Having said that, I have built tons of these workflows and can comment on a handful of things that nearly always come up. The first item to touch on is the general structure/outline of a VBA workflow.

This breakdown seems to always hold true.

Your VBA workflow will contain steps or sub steps that roughly be described as:

- Inputs
- Intermediate results
- Outputs

If your workflow is advanced enough to include a number of sub steps built from other steps, then you are likely to find that this breakdown applies within and across levels of your workflow. That is, the outputs of one step may very well be the inputs to another step. The intermediate result from one action will be the input for another.

When thinking in terms of these categories, there is a useful distinction to make that is somewhat unique to Excel programming: do your inputs and outputs exist in the Excel spreadsheet or only in the VBA code? This distinction is meaningful because it helps you think about how much of your workflow is the automation of otherwise human tasks (which could still be done by a human) vs. steps that are purely programmatic and could not be replicated by a human. Where this distinction is most likely to show up is when you are deciding where and how to perform a calculations. In theory, all of the Excel spreadsheet could be done in VBA via the `WorksheetFunction` object. Doing everything in VBA defeats a large part of the benefit

that comes from programming with VBA. It's easy to lose sight of this when you see a clean code-only solution to a problem, but realize that the greatest benefit to programming alongside Excel is that you have a powerful, human readable scratch pad that lives alongside your VBA.

As a comment, I have seen incredibly complicated workflows that involved detailed calculations of arrays that were donely exclusively in VBA. The math was fine and the results were generally useful. The problem was that there was no way to spot check a given result without debugging code. This makes it nearly impossibly for someone without VBA experience to validate your work. It also provides you job security, but ideally you'd gain security by other means.

A better marriage of VBA and Excel is to utilize Excel for all of the tasks it's great at: calculation, visual outputs, charting, page layouts and printing, and also the deep data oriented features (sorting, filtering, etc). Where VBA comes in handy, is wiring together all of these items into a coherent package that runs more efficiently than anything that a human alone could do. The best workflows typically take a very simply underlying spreadsheet and apply to a large number of items. In this way, you are able to spot check a single result, verify the formulas, and investigate an interesting result. You are also free to just hit go and have 10,000+ realists streamed into a table for consumption. IF you find yourself looking for all sorts of tricks to avoid using the underlying Excel model for your programming, I'd strongly encourage to just switch to a fully programmatic language that does not have the Excel UI. You will save yourself a

ton of headache. If you are only aware of VBA and looking to push the envelope in terms of performance, then that's an OK place to be. Just realize that there are better alternatives to Excel for high performance computing.

inputs

Back to the overall structure, there are inputs, outputs, and intermediate results. Depending on what you are doing, some of these aspects may just exist on/within the spreadsheet and be easy to overlook as an input or output. It's not until you wire up a more complicated workflow that you are forced to recognize the different pieces in a spreadsheet for what they are. On the input front, there are a handful of items that should trigger your thought of "input":

- A file that contains some data to be processed, filtered, etc.
- A couple of columns in a spreadsheet that need to be processed and then charted.
- 15 scattered cells that meet some criteria within a block of data
- THe contents of the clipboard from another program
- The formatting of a couple of cells

All of those items could be used as the input to a VBA workflow. Some of these items are odd to think about if you are coming from a noter programming environment. What does it mean for the formatting of

a cell to be an input? Well Excel provides you with a rich Object Model full of metadata about all of the various cells of data. That metadata can be as useful as actually structured data if there is a structure to it. I've seen it countless times where someone has methodically bolded all of the cells of intereste in a block of data. That bold format is as good as some field called `Important = True` which could then be processed in another language. Instead of that flag, you just check `Range.Format.Bold = True`. This of course relies on an implicit assumption about how the data is structured, but this is common in the Excel/VBA world.

Excel also has a very strong UI which makes it possible to immediately solicit user input in a way that is not easily replicated coming from other languages. Where this shows up most frequently is when you start using the `ActiveCell`, `ActiveWorkbook`, `Selection` and other objects which are dependent on user input. In a lot of other languages you have to spend a ton of time pointing the program to the correct file, or rows, or columns, or other items to process. In Excel, you leverage the fact that most people know how to select or activate items they want, and you can use that user input as an actual input to your VBA. This becomes quite powerful when you are building utility code that may be used across multiple workbooks. This becomes much harder in other languages where the idea of a "open file" is far less well defined. You certainly cannot query the selected cells in an R data table.

outputs

THe next item to hit are the outpost of a workflow. Very often, the outputs are obvious because you had some task to complete with VBA, and the outputs are simply the results of that task. Where things become more complicated is when you string together steps and the output of one becomes the input for the next. When that happens, you often have to decided what intermediate format is best for the transfer. You may or may not settle on a format that is easily human consumable. There are tradeoffs here that will be discussed later. The output of a workflow can be a number of things:

- A string, number, cell, row, column, or table of data that was processed by the VBA
- A chart
- A collection of shapes
- A worksheet that includes any of the items above
- A workbook that includes a number of constructed worksheets
- A change to the formatting of a number of cells
- A change to the properties of a Range, Worksheet or Workbook
- A new text file written to disk
- Some result output to the Clipboard

- Pages of physical paper if your VBA prints
- Some change to the filesystem or disk
- Some other program opened or run with specific parameters

This is a shortened list since the possibilities here are closer to endless. The idea however is that you can effect a large amount of change from VBA and so your possible outputs can be quite numerous. A typically workflow will accumulate a large number of these outputs individually and will then produce some final product which highlights some of those outputs.

intermediate results

When discussing intermediate results, it is generally best to limit your thoughts to whatever will live only in VBA. In that sense, the question of intermediate results is: what programming constructs can exist without the user ever seeing them? Sometimes you need to determine the unique items in a list to do some processing. Do you generate that list of unique items in Excel somewhere? Or, do you determine the unique items using VBA and then output some result which may or may not include the full list of unique items. If you are doing the former, Excel provides a nice `RemoveDuplicates` function which will replicate the `Data->Remove Duplicates` functionality. This works great if you want the user to see the final list of values. You can also use a `Dictionary` in VBA to only store the unique vaults from a list. In

this sense, the `Dictionary` represents an intermediate value that may not be shown to the user. You will make this decision several times before you realize that you are deciding whether or not something should exist in VBA only. Often times, the decision does not matter, but for certain workflows it can make a huge difference.

An example is a multi step process where you might want the user to verify the calculations so far and correct any errors. This can technically be done with VBA or Excel, but it is much easier to ask a user to verify an Excel spreadsheet than to debug the code and check `Locals`. If you need to do this verification step, then it makes a lot of sense to use an intermediate result that dumps back into Excel. In this sense, you've taken an intermediate result and converted it to an output. That output may or may not be modified by the user and it then becomes the input for the next step.

putting it all together

Having given a snapshot of the options for inputs and outputs, it's worth commenting generally on how they all fit together. Your goal should be to build a workflow that consists of steps that can all be described individually and possibly run on their own. Your task is then generating these individual steps and determining how to wire them together. The most common approach to building these workflows is that you start with some single task and then the scope expands as the analysis expands. You can build the ultimate

workhorse of a workflow initially, or you can adapt your code to the task as the task comes into view. Depending on where you're starting and the definition at the start, you will determine how complicated to make things at the start.

It is very common to start with a single, straight-through workflow and then build it out into Modules as the work expands. IN this way, you are constantly reevaluating the inputs nad output sof your program to build the smaller blocks which need these definitions. In my experience, nearly all VBA workflows will take shape in this process eventually. It's quite rate to build a complicate workflow once and for all. Generally you start simple and end up with a full featured application at the end.