**Macro-Enabled M-Line Drive Selection**

**How it All Works**

This document is mainly informational, and doesn’t serve much purpose besides answering some curiosities here and there. If any part of the process breaks, however, it may be useful to understand the flow of the code.This Drive Selection process is a lot more complicated than it seems, and sometimes the **Troubleshooting** guide can’t do much more than scratch the surface, even as comprehensive as I tried to make it. Here’s what happens when you press **Ctrl+m**:

* The code (written in Visual Basic) starts from within **DriveSelect.xlsm**, which exists purely as the “brains of the operation”. It contains around 2000 lines of code that all serve their own purpose during the Drive Selection process. As for why you have to open an Excel sheet to run this code instead of just simply clicking on an app, it’s far easier to contain a macro set within Office-related apps, therefore I chose not to write any outside code in other languages. Applications are not easily compatible with Visual Basic macros based in Excel. Improvements could certainly be made, but since this process is already cutting out around 90% of the time wasted on data entry and fan calculation, improvements may not be much more helpful.
* **DriveSelect.xlsm** will first redirect you to the PDF converter website. I decided to include this purely as a convenience, so that there isn’t too much work needed to be done outside of the program. In the background, Excel is set to wait **20 seconds** before throwing a new prompt up. There’s not a good way to be instantly directed back to the sheet as soon as the new file is downloaded: at least not without potentially throwing the program in an infinite loop if the file is never found. This way is just a little bit more work, but is much safer in the long run.
* **PDF2Go.com** is an external site used to convert PDFs into user-friendly versions. The URL extension /pdf-to-excel will, as you can guess, change the PDF into an Excel sheet. This level of conversion is far above my current coding knowledge, and is better suited for tools that already exists, rather than trying to re-invent the wheel, and because reading a PDF with OCR is ridiculously complex as opposed to simply converting to an Excel sheet.
* **DriveSelect.xlsm** will read in this Excel-exported version of a SMART order and enter all the standard data it can find on the sheet (part of the **Troubleshooting** guide dives into the expectations for what it should find, and what’s happening if it isn’t found). This data entry was accomplished through the use of Regular Expressions, which is a term familiar to those with experience in parsing. For those who aren’t familiar, Regular Expressions help break down extraneous details in data into the core, required information, which is what my code does. Visual Basic has a built-in Regular Expressions package that is easy to enable from within the code editor. Plenty of web resources exist if you ever have the interest to learn more about this.
  + The parsing portion of the code is actually astonishingly fast, for Visual Basic being such a high-level language (High-level languages tend to use up more computing power and take a while to execute, since they’re tailored more towards humans rather than machines). I chose to treat only Column 1 of the Excel/SMART sheet as having useful data (and all exported sheets so far follow this precedent). This eliminates the need to search the entire sheet, and speeds up the data parsing portion by a significant amount.
* After all the necessary information is grabbed off the exported SMART sheet for the current line, (including a prompt for your initials) the macros will open a workbook called **VFD\_Template**. It quickly makes a copy in your local **Downloads** folder called **Temporary.xlsm**, and does so to eliminate original template overwriting in the case of a fatal error. The macros use this **Temporary** sheet for the remainder of the entire order. The macros enter the data off the SMART sheet into all the appropriate cells within the template sheet.
  + This includes any VFD information. The macros are trained to search for the ACH580 VFD’s, so any other form of the VFD (yes, even 550’s) may not be an exact match. This is considered an accessory, and is not crucial to the accuracy of the overall Drive Selection, so it’s not the end of the world if one isn’t found/recognized
* While the data is being entered, the template is set up to continually call the fan calculation software each time a dependent cell is updated. For all units besides M140’s, the **Lau** fan modules are used within a file called **element\_DLL.dll**. **Comefri** is the sole supplier of M140 fans, so its file **aeobb32.dll** is used instead if the unit is an M140. This DLL file takes about four times as long as the Lau software, which is not something anyone can really change besides Comefri themselves. On very old computers, this can take several minutes to load, in the worst case
  + Why do I recalculate with each new data entry instead of once when all entries are finalized? It is best to leave the fan calculation software attached to the individual Drive Sheet in the case that the CFM, HP, RPM, or other information needs to be updated after the macros are done, and would impact the calculated blower RPM. This way, any update to those parameters would be instantly reflected in the **Calc RPM** field without having to call the DriveSelect macros again. (It won’t update the Drive Selection though)
* The macro set won’t do anything until the fan calculation modules finish their work. And these modules aren’t considered complete until the appropriate FLA and RPM information is entered on the template. Since the RPM is not something contained within the SMART sheet, a lookup to the **Motor\_BOM\_Tool** is required. The macros will open up this sheet and, knowing the type of motor that should be used, locate the motor entry on the sheet while finding the corresponding Baldor FLA and motor RPM. This data is entered into the template and the **Motor\_BOM\_Tool** is closed out, having served its only purpose for the current line
* The fan calculation modules run one last time, now given all the appropriate information. At this point, the calculated RPM for the blower should be accurate. It accounts for altitude, as well, making it all the more accurate. This should help to avoid hiccups in M-test.
* At this point, the sheet is ready for Drive Selection. The macros will open up **M\_Drives** as a lookup table, and filter through the thousands of entries in an attempt to find a suitable drive selection, or even a large grouping of acceptable ones within a suitable RPM range. It grabs this usable, filtered data from the sheet, and pastes this much smaller table into **Sheet2** of the Drive Sheet. This is useful in the (frequent) event a Drive Selection needs to be altered before being finalized.
* The macros use the template to step through each filtered entry, keeping track of the closest Drive RPM it can find without dipping below the necessary calculated RPM. This is used as the best-fit drive selection. In the event no good Drive Selection was found, or the ideal one wasn’t Engineering Verified, a message will alert you to this.
* The last major process this template will need to undergo is IVI selection, if even applicable. Since EVIs call for a much more intensive process, it isn’t feasible to include this within the process. If an IVI is requested by the customer, the template already has a built-in third sheet with a lookup table (named **IVI**), and the template very simply filters through the lookup table by the necessary parameters, which are already pulled from the SMART sheet anyway. In most cases, an acceptable (single) selection will be found, and a useful diagram exists within the sheet to help M-Line select and layout the specified springs.
  + If an IVI was not requested, the **IVI** sheet is deleted entirely. It isn’t necessary
* The template is now ready to be saved. It uses available cells on the template in order to format the name in accordance with the standard naming convention, leading by order number. In the rare case the order number wasn’t found (often due to a glitch in SMART formatting), the name given to the sheet will be “000 – ORDER NUMBER MISSING –“ followed by the line number. This way, unusual sheet names won’t be attempted during saving, and the error-prone sheet is effectively “pinned” to the top of the **Drive Selection Sheets** folder.
  + The template has a very useful save module built into cell **A40**. This can also be used, as explained in **Manual Entry Procedure**, in the case a sheet needs to be renamed. Please refer to that sheet before trying to use the module, so you know how it works.
* Once the template is changed to a usefully-named Drive Selection sheet, it is saved, closed out, and the process starts all over again for each line number in the current order. It makes individual sheets for each line number, since each line often contains different parameters
* During the entire process, each line’s Drive Sheet is continuously monitored by the code to detect any errors and/or blank entries. In the case that anything is missing (even a “Yes” in Engineering Verified), the code won’t suggest creating an email template automatically. There are three cases for what the code will do once the end of the order is reached:
  + Case 1 – All line numbers in the order had perfectly created Drive Sheets made. This is the ideal case, and the one with the least amount of follow-up work! A message will suggest you can create an email template immediately and with recipients in Sales, Operations, and M-test. It won’t send it automatically because you will still need to review the information
  + Case 2 – Some lines’ Drive Sheets in the order were perfectly created, but not all. This is somewhat less of an ideal case. You can’t email out partially complete orders, so no email template is suggested automatically. Feel free to open the imperfect sheets, review them, and save them over the old versions in the **Drive Selection Sheets** folder before choosing to create an email template with **Ctrl+e**.
  + Case 3 – No lines’ Drive Sheets in the order were perfectly created. This is the worst case, but often isn’t very bad. Almost all the relevant information will be present in the order, but a custom Drive may be required or a manual override of the Engineering Verification. It’s still easy to open the imperfect sheets, review them, and save them over the old versions in the **Drive Selection Sheets** folder before choosing to create an email template with **Ctrl+e**. On a side note, if you have a 90# motor, the macros won’t be useful in the slightest for such a custom component.
* After the final prompt given by the code, the **DriveSelect** sheet will reformat itself. It had previously copied all the information from the SMART-to-Excel sheet into its own worksheet. This was just for the sake of ease, and covers the extremely rare case the SMART information gets corrupted. Regardless, at any point the macro determines the process needs to be terminated (which includes after fatal error messages), it should attempt to wipe all the cells of their strange formatting, and put the two large prompts for macro execution back where they belong. The sheet saves itself so you don’t have to wonder whether to save it or not when closing it out