Dillon Franks

Final Project Write Up

Executive summary

I designed my project to be used by the accounting department where I work. I work at Progress Mfg., a manufacturing company that produces trailer hitches. I deal with invoicing customers and any other issues related to billing customers. I currently use many spreadsheets as part of my job. I decided to add VBA to two spreadsheets to automate the process and make it much quicker. The first spreadsheet imports customer credit card transactions and prints a statement. I wrote code that did all my previous steps with one click. The other spreadsheet is used to classify no charge invoices to their correct accounts. My program references data in another reference spreadsheet. My code analyzes that data and automatically classifies a many invoices as possible to the correct account. For those invoices that cannot be classified automatically, I use a user form to display relevant information from the reference spreadsheet. The user can then easily look at the data and classify it with their own intuition.

Implementation

I will first discuss the code used for the CC Auto Close spreadsheet, which is the one that imports credit card transactions data. I use this spreadsheet almost every day at work. I import multiple batches of online credit card transactions into the file, then print the information so it can be matched to invoices. I realized that VBA could automate the process of importing a text file, formatting the data, selecting the data, setting a print area, and printing the data. I created code that does all those tasks with a click of a macro button. First, due to the format of the cells, the data must be imported starting in cell A4, otherwise it won't print correctly. The code detects if the user tries running the program when A4 isn't selected. A message box displays that informs the user that A4 must be selected. If the user clicks cancel or exit, the program is terminated. If the user clicks ok, The Active cell will move to Cell A4 and the program and execute.

A CSV file is then imported. This is achieved by the program opening up a target CSV, copying the data, and then pasting the data into A4 of the worksheet. The CSV file is then closed. The data is selected starting in cell A4. Using the xldown function, data from A4 down to the end of data in that

column is selected. Then xlToRight selects the data to the right as well, thus all the data is selected. The print area is set to that selection and then it is printed. I created a macro button so that any user can easily run the program.

Another one of my job responsibilities is to make sure that all no-charge invoices are correctly classified. The COGS-Pivot Table spreadsheet contains a list of invoices that contain items that have been sent for no charge. The value column show the account that the invoice has been classified to. As you can see in figure 1, some invoices already have accounts assigned to them. Sales reps do this when they create the invoice. Sometimes, they do not remember to classify the account. This requires me to then go to the spreadsheet and examine all the N/A values. I use a reference spreadsheet called Customer Order and Invoice Lookup Tool to view information regarding the invoice that has an N/A value. The Customer Order and Invoice Lookup Tool pulls information from our Sage 300 database. I used SQL (figure 2) to join tables together so that I could obtain as much information as possible about every invoice. Previously, researching N/A values was sometimes a pain because I would have to search by the invoice number in the Customer Order and Invoice Lookup Tool, then look at many different columns to figure out how I should classify the invoice with the N/A value. It involved switching windows too often.

M
VALUE
#N/A
NC - Display
NC - Display
NC - Display
#N/A
NC - Display
NC - Display
#N/A
NC - Product Warranty (No RA#)
NC - Product Warranty (No RA#)
NC - Product Warranty (No RA#)
NC - Product Warranty (No RA#)
#N/A
#N/A
NC - Product Warranty (with RA#)

Command text:

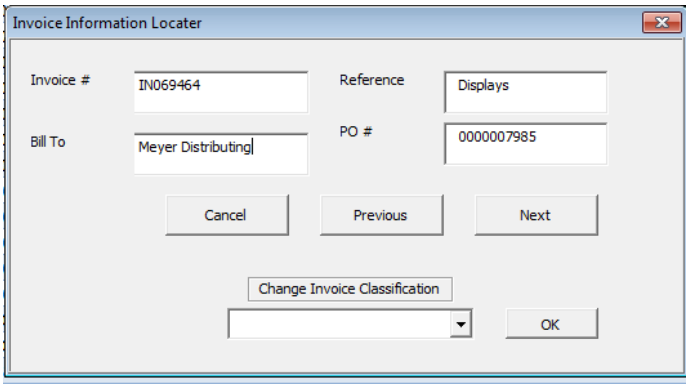
```
SELECT OEORDH.ORDUNIQ as 'UNIQUE ORD ID', OEORDH.AUDTDATE, OEORDH.CUSTOMER, OEORDH.BILNAME,
OEORDH.BILSTATE, OEORDH.BILZIP, OEORDH.SHPNAME, OEORDH.APPROVELMT, OEORDH.TERMS, OEORDH.REFERENCE,
OEORDH.ORDDATE, OEORDH.ORDFISCYR as 'ORD YEAR', OEORDH.VIADESC as 'SHIP VIA', OEORDH.LASTINVNUM as 'LAST
INV#', OEORDH.SALESPER1, OEORDH.ORDPAYTOT, OEORDH.SHIPDATE, OEORDH.INVDATE as 'INVOICE DATE',
OEORDH.INVAMTDUE, OEORDH.SHPADDR1, OEORDH.SHPADDR2, OEORDH.SHPCITY, OEORDH.SHPSTATE, OEORDH.SHPZIP,
OEORDH.SHPCOUNTRY, OEORDH.ORDNUMBER AS 'ORD #', OEORDH.PONUMBER, ARIBH.AUDTDATE, ARIBH.DATEINVC,
ARIBH.AMTINVCTOT, ARIBH.IDINVC as 'INVOICE#', ARIBH.ORDRNBR FROM ARIBH

RIGHT JOIN OEORDH ON ARIBH.ORDRNBR=OEORDH.ORDNUMBER
```

I have a few different parts of code in the COGS-Pivot Table spreadsheet that perform different functions. All the functions can be run easily from the Custom Macros ribbon. The open reference button opens the Customer Order and Invoice Lookup Tool (reference spreadsheet), which must be open for the Automate Warranties code and Invoice Information user form to work. Warranties are the most common account used to classify no charge invoices, so my program automates the process. First, the invoice number from the row that is selected is stored into a variable. The reference spreadsheet is activated, and the program searches for that invoice number in the reference spreadsheet. Once the invoice number is found, the cell that contains the invoice number that was found is set to a Range variable. By using the offset method on the Range variable, the program stores data from the PO Number column and Reference column from the corresponding row into two variables (see figure 3 for examples values that would be stored). The program looks to see if letter combinations such as

INVOICE#	BILNAME	CUSTOMER	PONUMBER	ORD MON	ORD D	ORD YEAR	ORI	REFERENCE
IN069760	RV's Northwest #1	RV09	NCWATMRA7325	10	14	2015	ORD57	RA0007325

“NCWA”,”RA”, and “RWK” are found in either of these two columns. The Instr function and an If statement are used to find if these combinations are contained in the stored variables. If one of those combinations is found such as they would be in the figure above, that means the no charge invoice should be classifed as a warranty [NC- Product Warranty (with RA#)]. The COGS-Pivot Table workbook is then activated. If the If statement is satisfied (one of the combinations is found), the code replaces N/A with “NC- Product Warranty” in the COGS-Pivot Table spreadsheet value column for the respective invoice number. The formula loops through every row where the classification for the invoice is N/A, and ignores the rows where the invoice is already classified.

The vast majority of N/A issues are solved with the warranty automation code. However, there are some issues that need human eyes to figure them out. The remaining invoices that display N/A as their classification still need to be examined. Many no-charge invoices are due to tradeshows, product giveaways, or displays for dealers. By using a user form, the relevant information about the selected invoice is shown so that someone can easily see the data to make the decision on how the zero invoice should be classified. One no longer has to go back and forth between spreadsheets looking for data and searching for values. It finds it automatically with a click of the button. This user form is run from the ribbon again under the Invoice Information button. The code that puts values in the form starts running when the form is initialized. Like before, the program looks up the invoice that is selected in the COGS-Pivot Table and finds it in the reference spreadsheet. Using the offset function, various values such as PO number and Bill To Name are pulled into the text boxes in the user form. The user can now see all information relevent to the invoice in one location, instead of having to look at multiple spreadsheets, as show in Figure 4. After looking at the data that is pulled into the spreadsheet, the user can choose from a drop down menu which account the invoice should be classifed as. I set up the user form to populate the accounts in the combo box when the user form is initialized. Once OK is clicked, the Values Column for the invoice is classified with the classification selected from the combo box. The user can then click next or previous to view the information for the next invoice. Even when the Values column is filtered so that only N/A values are shown, the next/previous button will skip over the hidden values and instead select the next visible cell. The user can look up information for each invoice for which there is an N/A value one by one and correct each classification until there are no N/A values



remaining. This program is effective because it eliminates me from having to search for a invoice number in the reference spreadsheet, look at all the data for the invoice, and then switch back to the COGS-Pivot Table spreadsheet. Switching between windows on a monitor can often get confusing, and that is now eliminated. I also don't have to type in the full classification account name now, all I have to do is select it from the drop down menu.

Learning and Improvement

One of the most important things that I learned from this project is that everything takes a lot more time to code than anticipated. However, I also learned that just about everything is possible in VBA. Almost every error can be fixed with sufficient research and creativity. Little differences between the needs for my project compared to prior in class examples made it so I had to find different ways to implement code that I thought I knew. For example, I thought it would be easy to switch to a different workbook in the code by activating it. However, every time the code tried switching back to the COGS-Pivot Table spreadsheet from the reference workbook in order to change the classification of an invoice, an out of range error would occur. I tried setting workbooks to variables as we learned in class, but even that did not work. Finally, I discovered that I could refer to each workbook as a number, which allowed me to switch between workbooks easily. I believe the problem was caused because when both files were open they were not in the same instance of excel. It took me 2-3 hours just to figure out that one problem. I learned that I had to be creative to solve the problems that occurred. I found solutions to some of the problems online but had to make many adaptations to have it work with my code. Another problem that I encountered was that the drop down list of classifications in the user form kept getting doubled each time I moved down to a new invoice. I figured out that this was happening because the code that added the values to the combo box was being run every time my code to save the value ran. I fixed the problem by only having the values added to the combo box when the form was initialized.

Also, the loop that ran the code until the active cell was empty had issues. Once the active cell was an #N/A value, an error would occur. The program would stop on the line of code that checked to see if the active cell was empty. Eventually I figured out that #N/A isn't really text, it's an error value. I changed the code to use the isEmpty method instead of the way that we learned in class and it worked. Also, at first I designed the code that searches the reference file to only run if the active cell contained the value "#N/A." That also gave an error and the code did not run properly. Learning from my prior mistake discussed above, I had an If statement check to see if the cell had an error, and if it did, then the

code that searched the reference sheet would run. In the future as I use this code at work, I hope to add code that will automatically classify invoices to accounts other than warranties. It was not necessary for the sample of values I used for this project, but it may be in the future. I would also like to make the user form interface more appealing graphically.

Assistance

Most of the assistance I received was for small sections of code. I usually resolved errors by searching about the error online, and then reading about how to fix it in the Mr. Excel forums. The most help I received was from the Microsoft help website for the section of code that imports data from a CSV file. A Mr. Excel forum gave me much needed help on how to select a cell below the current cell, without selecting hidden values caused by filtering the column.