Joshua Kirn
IS 520 Final Project Write-up
April 14, 2014

Contents

Executive Summary

My final project works with the user to quickly import, sort, and format .txt files. This was designed to simplify the work that I did during my internship at KPMG last summer and will continue to do when I return there for work after graduation. The example files that I have generated to accompany my project would probably take about an hour to import, sort, and format, and represent only a small portion of what I could expect to see on a given day. This program completes these mundane tasks with precision in a matter of seconds.

When I made my final project proposal, Professor Allen said he couldn't quite understand what I intended for my code to do. My work is somewhat unusual, and I think understanding my final project will be easier with an understanding of what I do for work.

As mentioned above, I work for KPMG's advisory practice in their Contract Compliance Services. We audit Software End-User License Agreements for our clients. For this, we receive information from our clients (usually IBM) about their clients (our "projects") that we will be reviewing. Our client gives us data about what software the project has purchased, what type of license they have for that software, and how many units of software they have purchased. We then have other KPMG associates or IT professionals from the client or the project conduct reviews of the information systems at each of the project's sites to collect information about the project's actual software usage.

The reports from the client and the project are sent to me, where I have to import the data into Excel, sort it to the correct tabs, format it correctly, analyze it, and ultimately draw conclusions about how much money the project owes our client. The issue is that frequently, these jobs will involve hundreds of reports from project sites, each with anywhere from one to hundreds of records. Each time new reports are brought in, I have to redo the process, including needing insert rows, putting each bit of data into the correct cells, reformatting the sheets, and showing correct subtotals for each software group (when you replace subtotals over and over again, you realize the subtotal function is such a bigger burden than it needs to be).

The importing, sorting, and formatting of data takes a completely unreasonable amount of time relative to the amount of time that is spent actually analyzing the data and drawing conclusions. This project streamlines the importing, sorting and formatting of data to make the process much quicker, especially in regards to automatically formatting any data additions to already-in-progress files.

Implementation Documentation

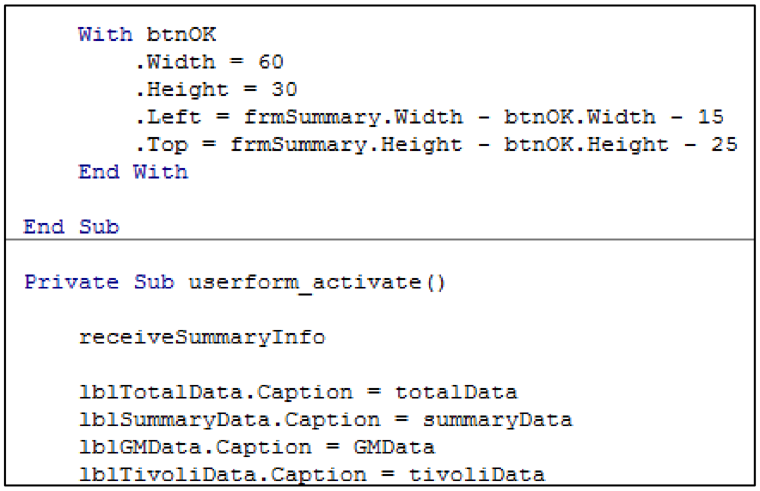
User Interface

I recommend following the tutorial I wrote below as supplementary information for a short and comprehensive review of the user experience. Upon opening the data input user form, the user will be given two options: to begin a new workbook or to add data to the file already in existence.

By starting a new workbook, the program will remove all information existing in the workbook, create all necessary sheets, and format them. The user will then be required to input the data that will be inserted in the header for each sheet. Next, the user can browse for the file to be input or enter the path directly and indicate to the user form whether the report belongs to the project or the client. At this point the user can input the data, and the program will run all applicable macros, and then return a summary statistics sheet showing how many lines of data were brought in and to which sheet they were sorted. Finally, the user is given the option to return to the input data form to input another file.

By adding data to an existing workbook, the user forms and effects are similar in most respects with a few significant differences. Most importantly, the existing workbook will not be cleared and reformatted. Additionally, the user can elect to skip adding data to the sheet headers, or can choose to change any number of the 4 fields if some of the information has changed or was spelled incorrectly at any point.

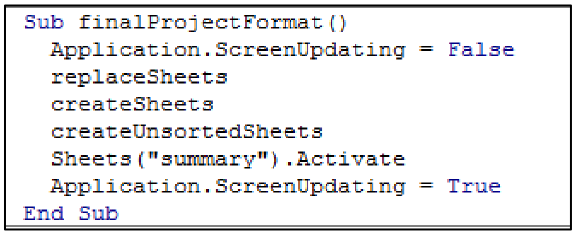
Coding



```
    With btnOK
        .Width = 60
        .Height = 30
        .Left = frmSummary.Width - btnOK.Width - 15
        .Top = frmSummary.Height - btnOK.Height - 25
    End With

End Sub

Private Sub userform_activate()

    receiveSummaryInfo

    lblTotalData.Caption = totalData
    lblSummaryData.Caption = summaryData
    lblGMData.Caption = GMData
    lblTivoliData.Caption = tivoliData
```

Part of the summary statistics form code, showing relative placement of button and calling of variables from other modules

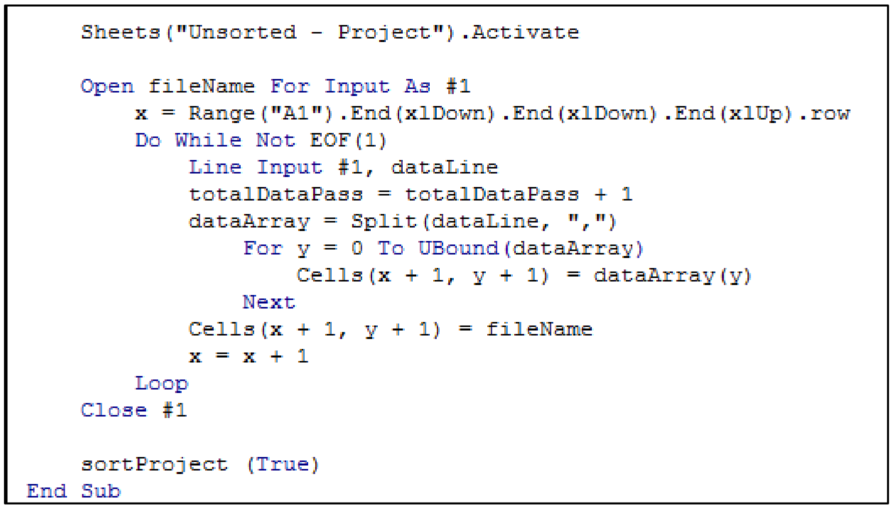
The user forms were coded to all appear relatively similar, with all differences relating to the nature of the objects that exist on the forms. For example, all "OK," "Cancel," or equivalent buttons will sit in the same spot relative to the bottom-right corner of the form. The user forms also pass to and call any data from user forms to implement the correct type of sorting and create the summary statistics form.



```
Sub finalProjectFormat()
  Application.ScreenUpdating = False
  replaceSheets
  createSheets
  createUnsortedSheets
  Sheets("summary").Activate
  Application.ScreenUpdating = True
End Sub
```

The main sub in the format module

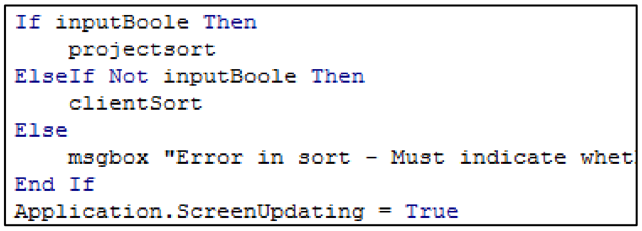
The formatting module deletes all files in the workbook and replaces them with fresh worksheets that are formatted according to KPMG standards (please don't penalize me for the ugly yellow color scheme! It's not my fault!). It produces the sheets necessary for the work to be done—the summary sheet as well as one sheet for each family of our client's (in this case IBM's) software. The formatting module also produces the "Unsorted" sheets which are relevant to error detection, as discussed below.



```
    Sheets("Unsorted - Project").Activate

    Open fileName For Input As #1
        x = Range("A1").End(xlDown).End(xlDown).End(xlUp).row
        Do While Not EOF(1)
            Line Input #1, dataLine
            totalDataPass = totalDataPass + 1
            dataArray = Split(dataLine, ",")
                For y = 0 To UBound(dataArray)
                    Cells(x + 1, y + 1) = dataArray(y)
                Next
            Cells(x + 1, y + 1) = fileName
            x = x + 1
        Loop
    Close #1

    sortProject (True)
End Sub
```

The main portion of the upload code for a project file. Note the last line where a Boolean passes to the sort sub, indicating to sort the new data as a project file.

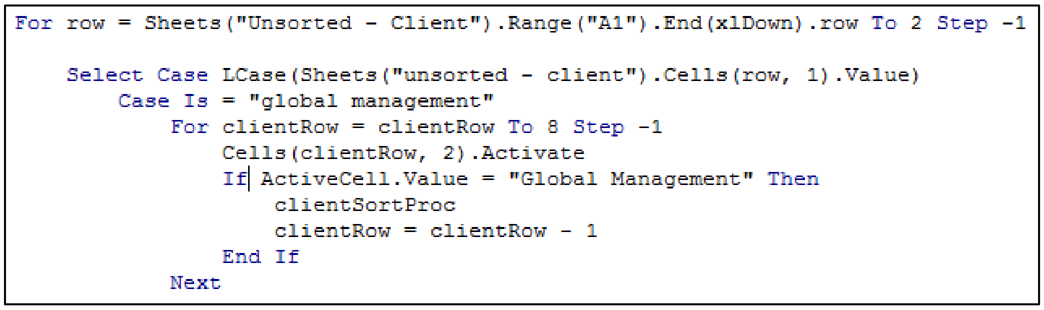
The upload module includes to macros that are almost identical with two exceptions related to whether the .txt file is indicated to be a client or a project file. These relate to which Unsorted tab the macro will print to as well as which sorting procedure will be followed. Essentially, the upload macro first ensures that the Unsorted sheet exists, and then reads the .txt file line-by-line, using comma delimiters to break the line into separate cells, and inputting each line to the applicable Unsorted sheet. Last of all, it will input the path for the .txt file to allow users to trace the data back to their source if it later needs to be reviewed.



```
If inputBoole Then
    projectsort
ElseIf Not inputBoole Then
    clientSort
Else
    msgbox "Error in sort - Must indicate whet
End If
Application.ScreenUpdating = True
```

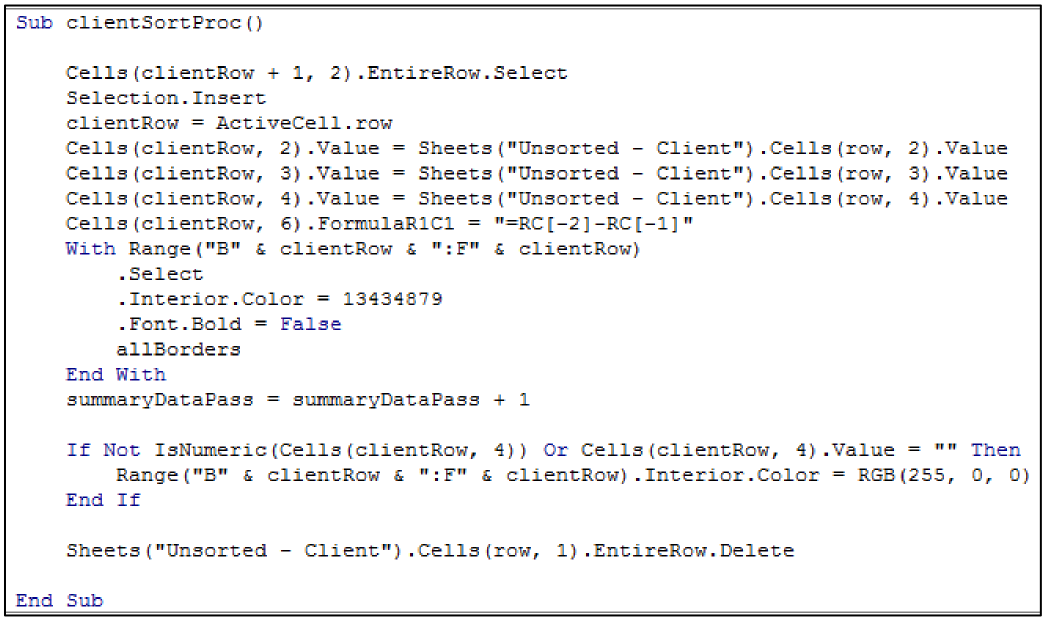
The main sub in the sorting module, which receives the Boolean passed by the upload sub to select the correct procedure to follow

The sort module reads the data which has now been brought into Excel and decides where it should go. This is the most crucial part of the project. In order to complete this, the code needs to review the Unsorted data for errors, find the correct sheet for the data, find the row where the data should be placed, input each cell from the Unsorted sheet where it belongs, format the finished row, check for completeness of data, and delete the row of data from the Unsorted sheet. This was done through several layers of loops and a few select case or if/then statements. The screenshots on the next page show a snip of how this was done for client files.



```
For row = Sheets("Unsorted - Client").Range("A1").End(xlDown).row To 2 Step -1

    Select Case LCase(Sheets("unsorted - client").Cells(row, 1).Value)
        Case Is = "global management"
            For clientRow = clientRow To 8 Step -1
                Cells(clientRow, 2).Activate
                If ActiveCell.Value = "Global Management" Then
                    clientSortProc
                    clientRow = clientRow - 1
                End If
            Next
```

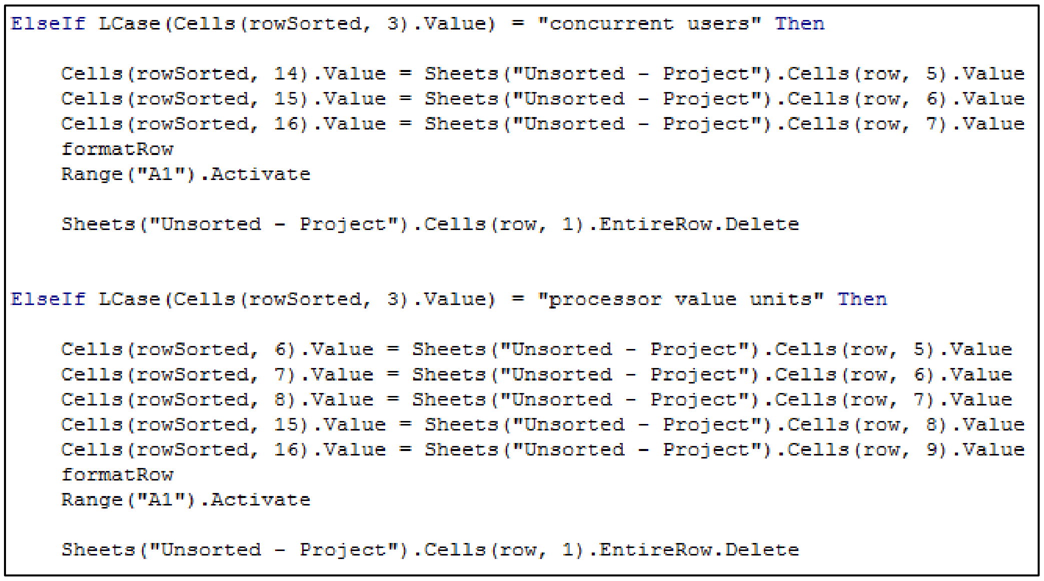
Part of the client sort sub, finding the correct row on the summary sheet before sorting the data there.



```
Sub clientSortProc()

    Cells(clientRow + 1, 2).EntireRow.Select
    Selection.Insert
    clientRow = ActiveCell.row
    Cells(clientRow, 2).Value = Sheets("Unsorted - Client").Cells(row, 2).Value
    Cells(clientRow, 3).Value = Sheets("Unsorted - Client").Cells(row, 3).Value
    Cells(clientRow, 4).Value = Sheets("Unsorted - Client").Cells(row, 4).Value
    Cells(clientRow, 6).FormulaR1C1 = "=RC[-2]-RC[-1]"
    With Range("B" & clientRow & ":F" & clientRow)
        .Select
        .Interior.Color = 13434879
        .Font.Bold = False
        allBorders
    End With
    summaryDataPass = summaryDataPass + 1

    If Not IsNumeric(Cells(clientRow, 4)) Or Cells(clientRow, 4).Value = "" Then
        Range("B" & clientRow & ":F" & clientRow).Interior.Color = RGB(255, 0, 0)
    End If

    Sheets("Unsorted - Client").Cells(row, 1).EntireRow.Delete

End Sub
```

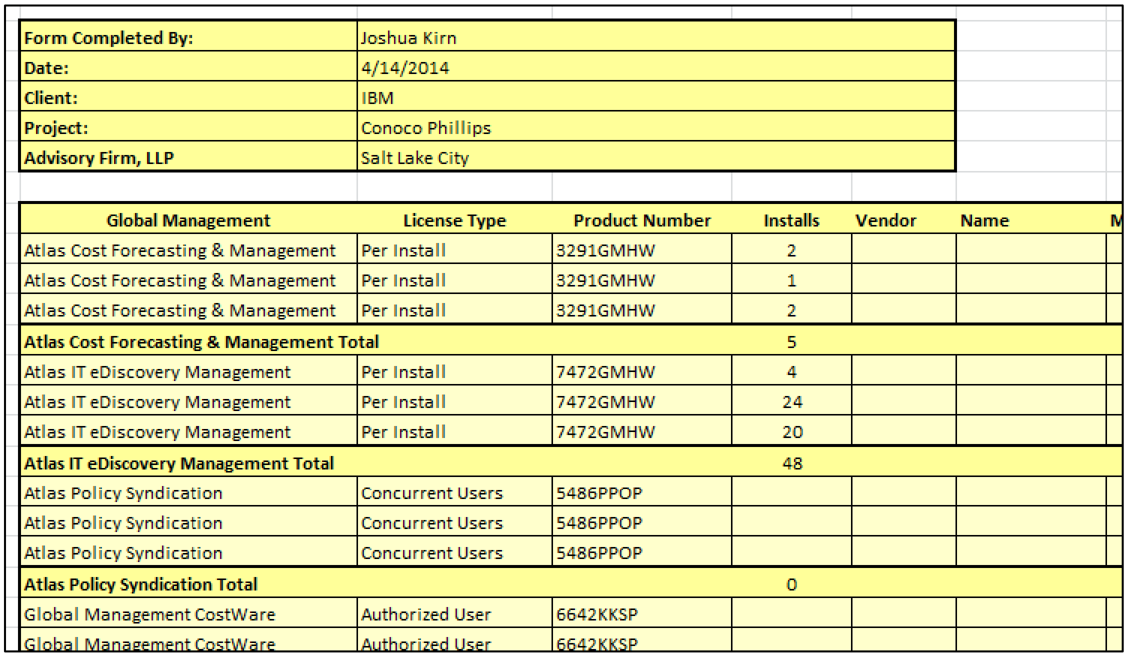
The next step; sorting and formatting the data that was input into the Summary sheet, detecting whether the row appears to contain incomplete data and highlighting the row red if it does, and deleting the now-sorted data from the Unsorted sheet.

The project files were sorted in a conceptually similar way, adjusted for the use of multiple sheets and the requirement that certain columns received data while others were left blank, depending upon the licensing type. Project files also have the additional requirement to check for a valid license type to sort correctly and to add subtotals once the entire file has been sorted. The next page shows a few screenshots demonstrating how these work.



```
ElseIf LCase(Cells(rowSorted, 3).Value) = "concurrent users" Then

    Cells(rowSorted, 14).Value = Sheets("Unsorted - Project").Cells(row, 5).Value
    Cells(rowSorted, 15).Value = Sheets("Unsorted - Project").Cells(row, 6).Value
    Cells(rowSorted, 16).Value = Sheets("Unsorted - Project").Cells(row, 7).Value
    formatRow
    Range("A1").Activate

    Sheets("Unsorted - Project").Cells(row, 1).EntireRow.Delete


ElseIf LCase(Cells(rowSorted, 3).Value) = "processor value units" Then

    Cells(rowSorted, 6).Value = Sheets("Unsorted - Project").Cells(row, 5).Value
    Cells(rowSorted, 7).Value = Sheets("Unsorted - Project").Cells(row, 6).Value
    Cells(rowSorted, 8).Value = Sheets("Unsorted - Project").Cells(row, 7).Value
    Cells(rowSorted, 15).Value = Sheets("Unsorted - Project").Cells(row, 8).Value
    Cells(rowSorted, 16).Value = Sheets("Unsorted - Project").Cells(row, 9).Value
    formatRow
    Range("A1").Activate

    Sheets("Unsorted - Project").Cells(row, 1).EntireRow.Delete
```

The most drastic difference between sorting client and project files; project files will bring data to different columns based on the license type. This means that each row should contain data relevant for one of the four license types with the other license type columns left blank.



Form Completed By:	Joshua Kirn
Date:	4/14/2014
Client:	IBM
Project:	Conoco Phillips
Advisory Firm, LLP	Salt Lake City

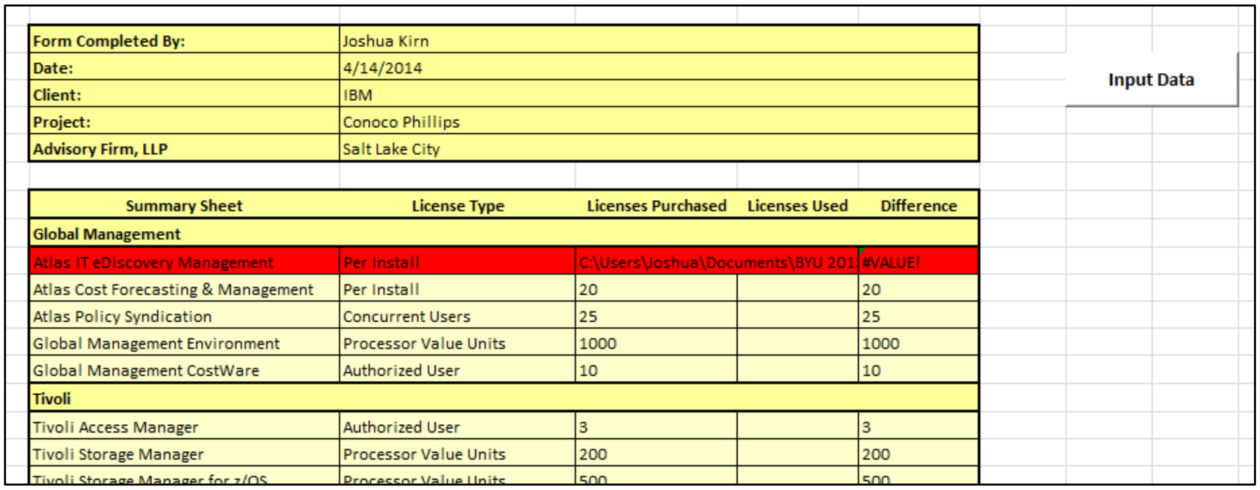
Global Management	License Type	Product Number	Installs	Vendor	Name	M
Atlas Cost Forecasting & Management	Per Install	3291GMHW	2			
Atlas Cost Forecasting & Management	Per Install	3291GMHW	1			
Atlas Cost Forecasting & Management	Per Install	3291GMHW	2			
Atlas Cost Forecasting & Management Total			5			
Atlas IT eDiscovery Management	Per Install	7472GMHW	4			
Atlas IT eDiscovery Management	Per Install	7472GMHW	24			
Atlas IT eDiscovery Management	Per Install	7472GMHW	20			
Atlas IT eDiscovery Management Total			48			
Atlas Policy Syndication	Concurrent Users	5486PPOP				
Atlas Policy Syndication	Concurrent Users	5486PPOP				
Atlas Policy Syndication	Concurrent Users	5486PPOP				
Atlas Policy Syndication Total			0			
Global Management CostWare	Authorized User	6642KKSP				
Global Management CostWare	Authorized User	6642KKSP				

A portion of the Global Management sheet after a project sort; notice that the "Installs" column receives data for software with the "Per Install" license type but not for the "Concurrent Users" or "Authorized User" license types. These would receive data in columns N and M, respectively.

Error Detection

The .txt files rarely contain misspelled or incorrect data since the reports go through software at the project site; the issue that we face much more often is incomplete data. As you can see in the workbook, I have intentionally left a few lines from the .txt files incomplete—they are the ones that are highlighted in a bright red. My code should automatically detect if something seems incomplete about the data after the import and sort have been completed so we can follow up with the project site to get whatever data was left out. It is better to sort these data and have them be temporarily incomplete than it would be to leave them in the unsorted data sheets.

The unsorted data sheets are a helpful control when inputting data also. In practice, the files that get input contain several lines of irrelevant data for each line that is useful. The unsorted data sheet is designed to be quickly examined by the user after they have finished their sorts to review for any information that may have been missed by the sorter for whatever reason before deleting all of the junk on the sheet. In addition, you are unable to import any file that is not a .txt extension; however, if you try to import an irrelevant .txt file to the workbook, it will gather all of the data in the unsorted data sheet to be quickly disposed of.



Form Completed By:	Joshua Kirn
Date:	4/14/2014
Client:	IBM
Project:	Conoco Phillips
Advisory Firm, LLP	Salt Lake City

Summary Sheet	**License Type**	**Licenses Purchased**	**Licenses Used**	**Difference**
Global Management				
Atlas IT eDiscovery Management	Per Install	C:\Users\Joshua\Documents\BYU 201		#VALUE!
Atlas Cost Forecasting & Management	Per Install	20		20
Atlas Policy Syndication	Concurrent Users	25		25
Global Management Environment	Processor Value Units	1000		1000
Global Management CostWare	Authorized User	10		10
Tivoli				
Tivoli Access Manager	Authorized User	3		3
Tivoli Storage Manager	Processor Value Units	200		200
Tivoli Storage Manager for z/OS	Processor Value Units	500		500

The top of the Summary sheet, showing one red line where I deliberately left incomplete data prior to the data input.

The red lines also appear if the user makes an error by selecting a client.txt file but tells the program that the file belongs to the project, or vice versa. This mistake would cause all of the data to be imported and sorted, but all of it would appear in red, allowing the user to immediately recognize the error. Other bulletproofing, including invalid responses in the user forms that could lead to program failures, you can examine on your own if you would like to.

Discussion of Learning and Conceptual Difficulties Encountered

Some aspects of the project were more simple than I had expected, while some were very difficult for me. When I originally submitted my project proposal, I was under the impression that importing data from .txt files would not be possible due to my misunderstanding of a comment made by Professor Allen in class. Based on the work we did on the read a file assignment and the database manipulation and user forms projects I was able to import the data. That was probably the only part of the project that took far less time than I had expected.

The part that was much more difficult than I had anticipated was the complexity of using multiple forms, particularly in needing to pass values between my forms, macros, and looping the forms. It seemed that whenever I found a bug, I would have to cycle through all of my forms and modules to fix it everywhere, which was monotonous and frustrating.

The best example of this is the statistical summary user form that appears after the upload and sort is completed. One of the key reasons that this was difficult was that I added it to my final project to increase the scope after I realized that importing data was far easier than I had expected. Because of this, I implemented this form after the majority of my project was completed, and so I had to go through everything else to get it to work properly. I also faced difficulty with this form in getting the numbers to add up correctly when I imported multiple files at once. Between this and the other user forms, I spent several hours online researching how to get them to work properly.

Did it meet my own expectations?

One of the keys I focused on was getting my project to look extremely polished and professional. Because of this I made an effort to make everything as consistent as possible throughout the user experience, including form and button sizes, button locations within the forms, colors, borders, and column widths throughout the workbook. While this wasn't always complex, it took a lot of diligence to get everything just how I wanted it to look—and I am very pleased with the user interface.

There is one point where there is a formatting error that I debugged for a few hours without being able to figure out the issue. After following the tutorial that I outlined below, you may notice one row that should have all of the borders but instead has only the borders shared by the rows above and below it. However, the row has all of the other formatting that it is supposed to have—I cannot figure out why it seems to be skipping the line that commands it to be formatted with all of the borders. I faced several difficulties in getting each row to automatically format as the sort progressed, and everything else is exactly how I want it to be, but I just can't get rid of this one last bug.

The other key that I focused on was the precision of the sorting process. Getting everything to go to the correct location and highlighting potentially incomplete or incorrect data was vital. Ultimately the whole project can look like a million bucks, but if it doesn't work, it has no worth. Planning out the flow of data before starting to write my code was very beneficial in getting the data to sort correctly. Of course, I still had issues with the code, but I feel like I would have spent hours longer had I not planned out the flow beforehand.

Why did I not include other features?

A few things that I considered adding to my final project that I ultimately chose against include dynamic file import (like we did in the database manipulation project), email alerts upon completion of the data import and sort, and a customized ribbon. The one feature that I truly wanted to add to the project was the customized ribbon. I thought that it would be really cool and add a touch of professionalism to the project, on top of offering a convenient way to access the user forms and macros that I had created. I simply ran out of time available to make this feasible, so I compensated by creating the user form that opens up when the workbook is opened and adding the Input Data button on the summary sheet to allow convenient access to the user forms.

The option to browse for files is better than the dynamic file import in this instance because of the nature of the data sorting and timing of reports. Since some files must be sorted as client files and others are sorted as project files. Since the .txt files are not dramatically different, I have to specify how each file will be sorted. I am sure this problem can be remedied; the more relevant issue is that some reports are received when the workbook is already in progress. By having a dynamic file import I could be double-counting files that have already been imported which will destroy the integrity of the data. The bit of time required to loop through the user forms is preferable to needing to wait for all of the reports (which can come in over the course of several months) before beginning analysis.

Finally, my decision to not send email alerts was based on practicality. Even the largest files that I will deal with will not take long for the VBA to sort through. At least 99% of the time it would take longer for the user to input their email address than it would take to wait for the program to run.

How to improve the project

Since I am a novice programmer I am certain that my code is not as efficient as it could be. As it is, my final project does not likely use too much processor power as-is, performance could probably still be improved.

There is one spot in my program where I feel that a user mistake could lead to a crucial error, and that is the possibility that a user selects to make a new file instead of adding data to an existing file at the main data input user form. This would particularly be an issue if the user has already input several reports, as it will delete any information that currently exists on the workbook. I currently have a confirmation form that makes sure the user intends to delete any data on the worksheets, and the only thing I can think of is adding more confirmation forms that explicitly tell the user the consequences of making a new file (are you sure??? ARE YOU SURE???). If I could create a preventive control that restricts the user's ability to delete everything in the workbook, this would be a better program.

I look forward to showing this project to my manager in a few weeks and receiving suggestions from him as to how I can add helpful features to the project.

On a personal note, thank you for your instruction this semester and for reviewing my project. I have really enjoyed the course, and I look forward to finding ways to improve and use my VBA skills in the future.

Assistance

I did not receive any help from any other people in completing this project. Any difficulties were addressed by consulting the course textbook or online VBA resources.

Supplementary – Tutorial for Grader

This is a basic set of instructions that should give you a quick, easy way to run through my workbook and see its features completely. This is not technically part of the write-up, but it should allow you to quickly understand my final project by following the steps.

1. Upon opening the workbook, a user form will appear allowing you to input data or proceed to the workbook. Select the Input Data button to bring up the next form.
2. Select "This will be a new file" and press OK. Press Continue on the following form; this will initialize the formatting of the workbook.
3. Insert a value into each text box and press OK. You need to input some value into each box to proceed. There is no check that ensures you are inputting the correct information, but incorrect values at this spot will not have any major effect on the workbook and I assume professionals can correctly identify and spell their names and office locations.
4. Select Browse and locate the file client.txt that I included in my submission folder. Select client and OK. Later I will discuss what occurs if the user enters an incorrect prompt at this point. This will upload the information from the client.txt file and sort all of the data to the appropriate rows within the workbook, segregating any error lines for human review.
5. Press OK on the summary statistics form and press Yes when asked if you would like to upload another file on the next form.
6. This time select to add data to an existing file and press OK. Note that this will not format the workbook as it did when "This will be a new file" was chosen.
7. Input data into one of the text boxes and press OK. This form will update the headers for only the information that you elect to change by ignoring any empty strings in the text boxes.
8. Select Browse and locate the file Project1.txt that I included in my submission folder. Select project and OK. Again, this will upload the information from the client.txt file and sort all of the data to the appropriate rows within the workbook, segregating any error lines for human review.
9. Press OK on the summary statistics form and press Yes when asked if you would like to upload another file on the next form.
10. Select to add data to an existing file again and press OK. Press Keep Existing Information on the next form.

11. Select Browse and locate the file Project2.txt that I included in my submission folder. Select Project and OK.
12. Press OK on the summary statistics form and press No when asked if you would like to upload another file on the next form.
13. You can skim through the workbook at this point to review the work that was done through VBA as you worked through the forms. Below I will explain what the purpose of this exercise is and how the programming meets the desired functionality of the workbook.
14. If you want to attempt to "break" the system, you can press the Input Data button that was placed on the Summary sheet and access the forms again. You should not ever notice any Debug prompts. Please inform me of any issues that you find.