Cash refund process at BYU could take days until decision is finalized mainly because refund request is written down on a hard copy paper, and the paper goes to several people for tedious checks and evaluations. Recently, BYU adopted a program called Informatica to automate most of the tedious steps of the refunding process. The program works fine besides the fact that it does not support files that have .xls (excel) extension. When a new refund request is submitted in .xls file, people have to manually convert .xls file to .csv file to run the file in the program. I was asked to write a batch system with VBA to convert however many .xls files to .csv files with few clicks of mouse.

As a proposal, I committed to add the following user cases to the batch system:

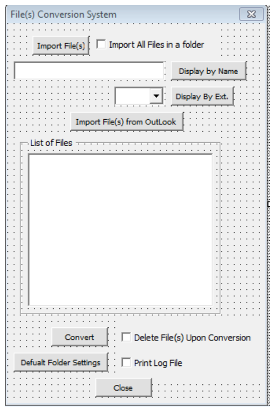
1. Employee sees a friendly GUI that takes almost no train to use.
2. Employee selects a list of files or a file from a folder and displays the file(s) in the GUI.
3. Employee selects a specific file or all of the files listed in the GUI for conversion
4. Program produces a log after the conversion process is finished.
5. Employee is alerted when the selected files aren't converted properly.
6. Program downloads attachments of emails and lists the files in the GUID and put them in the default folder.
7. Employee searches a specific file from the list of the uploaded files by typing in the input box.

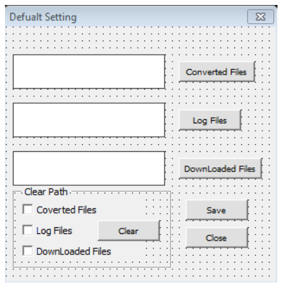
Converting each .xls file to .csv is very tedious and is definitely susceptible to human errors. Using this program will enhance the work time and the productivity.

Graphic User Interface

User Case: Employee sees a friendly GUI that takes almost no train to use.

The following screen shots are the interfaces of my program. Caption of each button is self-explanatory, and components of the interfaces are aligned and organized according to their relevance. Importing and searching functions are placed in the top of the interface whereas finishing functions such as conversion process are placed in the bottom that the program is neatly designed for employee to work from top to bottom.


File(s) Conversion System
Import File(s)
Import All Files in a folder
Display by Name
Display By Ext.
Import File(s) from OutLook
List of Files
Convert
Delete File(s) Upon Conversion
Defualt Folder Settings
Print Log File
Close



I did not particularly have a problem creating this interface, but I had to spend time to think about the arrangement and the organization of each component's placement in the interface. I also had to come up with names that are unique but could be easily remembered for each component so that I can refer back any of the component without looking at the codes.

Displaying File(s) in the Interface.

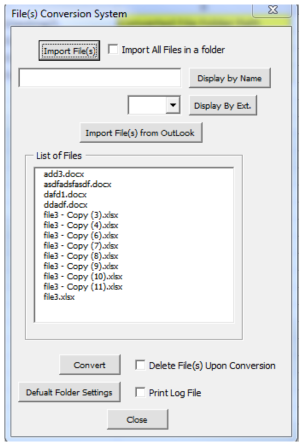
User Case: Employee selects a list of files or a file from a folder and displays the file(s) in the GUI.

The first step to run this program is to decide what files are going to be converted. Any types of files could be uploaded in the interface, but only files that have .xls(m) extension will be converted to .csv file and the rest will be thrown away. I could add few lines of code to support other file extensions but the scope of this project is just to have .xls file converted to .csv file. I will talk more about the codes for conversion process later.

Files could be uploaded in the interface by simply double clicking Import File(s) button; before clicking this button, employee needs to decide whether to have the program to upload all files in a single directory or to upload only the handpicked files. If ☐ Import All Files in a folder is checked, all files in the selected directory will be uploaded in the program.

```
lstFile.Clear

If singleAll.Value = True Then

    Set fd = Application.FileDialog(msoFileDialogFolderPicker)
    fd.Show
    path = fd.SelectedItems(1) & "₩"
    drt = Dir(path, vbNormal)

    Do While drt <> ""
        lstFile.AddItem drt
        drt = Dir
    Loop

Else

    Set fd = Application.FileDialog(msoFileDialogFilePicker)
    fd.Show
    path = Left(fd.SelectedItems(1), InStrRev(fd.SelectedItems(1), "₩"))

    For Each file In fd.SelectedItems
        lstFile.AddItem Right(file, Len(file) - InStrRev(file, "₩"))
    Next

End If
```

The block of codes above is what I wrote to get the job done. After the codes are ran, the interface will display files' name and extension like the screen shot below.


File(s) Conversion System
Import File(s)
Import All Files in a folder
Display by Name
Display By Ext.
Import File(s) from OutLook
List of Files
Convert
Delete File(s) Upon Conversion
Defualt Folder Settings
Print Log File
Close

add3.docx
asdfadsfasdf.docx
dafd1.docx
ddadf.docx
file3 - Copy (3).xlsx
file3 - Copy (4).xlsx
file3 - Copy (6).xlsx
file3 - Copy (7).xlsx
file3 - Copy (8).xlsx
file3 - Copy (9).xlsx
file3 - Copy (10).xlsx
file3 - Copy (11).xlsx
file3.xlsx

When the files are selected, I didn't want the program to display path; therefore, I had to think of a way to truncate the full path of each file to just the file's name and extension, and I also had to make sure that I can recall the path when the files are going through the conversion process. After searching Google for a while, I found out that there is inverse version of InStr function which is called InStrRev. What this function does is that it finds the first occurrence of the search key character, in the case above "\", and counts the position of the key character from right to left instead of left to right. This is very useful because I recognize that the entire file name starts right after the last back slash appears.

The program will not allow employee to upload files on top of already uploaded files; I've written the program in this way because I was given an instruction that all of the files that required to be converted will be stored in the same directory. When the program detects files that are already converted in the past and is attempted to be converted again, Excel will prompt a warning sign and asks if the employee wants to override the file.

The hardest part of the functionality to upload files was to find a replacement of "FileSearch" property. When I researched on how excel handles files from Google, majority of the VBA writers were referring "FileSearch" property; as a matter of fact, I believe it's the easiest way to handle files; but, unfortunately, Bill Gates decided not to support the property in any upper version of Excel 2003. After spending a good amount of time, I found out about FileDialog property. Using this property, it was easy to get the full path of each file that I selected; but getting all files in a single directory by just double clicking the directory alone, I had to think a different approach. The solution of this problem was to use "Dir" function. If I refer back to the block of code listed above, Dir function gets each file in a directory without having to know what is in the directory.

When I clicked the X button in the upper right hand corner of the interface, the program crashed immediately. I had to restart the program each time when I did that. I didn't like to have the program restarted when the x button is clicked. I wanted to run the program as smooth as possible. I solved this problem by using the following block of code.

```
On Error GoTo ErrHandler

'(Codes come here)

Exit Sub
ErrHandler:

'(Error handler codes come here)
```

If any of the codes between On Error GoTo ErrHandler and Exit Sub cause errors while the program is up and running, the errors will be captured and codes below ErroHandler: will be executed. In the case above, I instruct the program to just exit the sub.

File Conversion

User Case: Employee selects a specific file or all of the files listed in the GUI for conversion

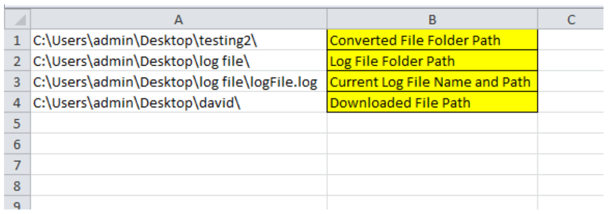
Since converting the files is the main purpose of this program, lots of things will be happened when employee double clicks on Convert button. First, I want to employee to be able to decide whether to keep the original files after conversion or not because I've heard many times that when the .xls files are converted, the files will no longer be used, but I still wanted to give employee an option to keep the file; therefore, I added a ☐ Delete File(s) Upon Conversion checkbox. When the box is checked the block of codes below will be executed after the file(s) are converted.

```
If DelFiles = True Then

    SetAttr tempPath, vbNormal
    tempFile.Close savechanges:=False
    Kill tempPath

Else

    tempFile.Close savechanges:=False

End If
```

One thing that I had to make sure before the files are deleted was that the files aren't read only. Sometimes Excel automatically sets files to be read only when the files are open. Second, I want to allow employee to decide either converting all files in the list or a specific file. The following block of code will get the job done. If no file is selected lstFile.ListIndex will return -1 value.

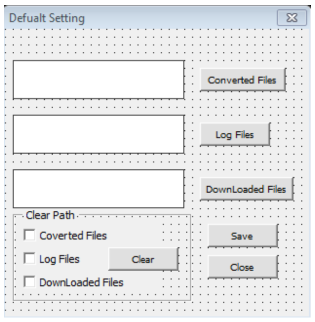
```
If lstFile.ListIndex <> -1 Then
    '(If a speicific file is selected)
Else
    '(Convert all files in the list)
End If
```

I wanted employee to be able to manually specify where the converted files are going to be stored, and I also wanted the files to be stored automatically to the default folder; the hardest part of this problem was to have the program to remember the default directory path because once the program is closed all values used during the process will be wiped out and start new when the program is restarted. Since Excel file can contain values in a spreadsheet, I made a sheet like below and stored the full path of the default directory.

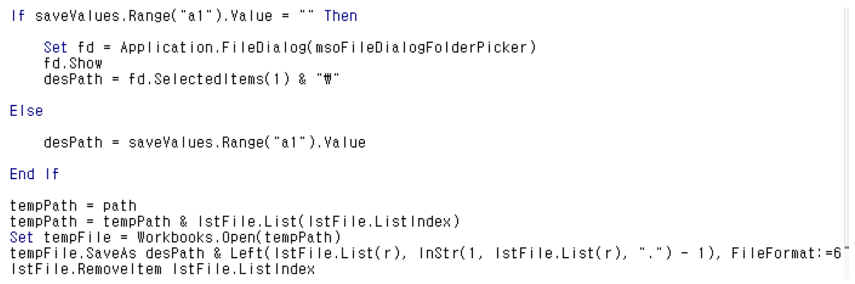


	A	B	C
1	C:\Users\admin\Desktop\testing2\	Converted File Folder Path	
2	C:\Users\admin\Desktop\log file\	Log File Folder Path	
3	C:\Users\admin\Desktop\log file\logFile.log	Current Log File Name and Path	
4	C:\Users\admin\Desktop\david\	Downloaded File Path	
5			
6			
7			
8			

Instead of making employee write down full path of the default directory directly in the spreadsheet, I made another interface below to have employee able to pick a directory. When employee clicks any of the buttons next to white boxes, a browse screen will appear. After reconfiguring the paths, employee can either click Save button or Close button. If Save is clicked then every reconfiguration made will be saved to the workbook, whereas Close just closes the interface. I used ActiveWorkbook.save to save the value and Unload me to close the interface.

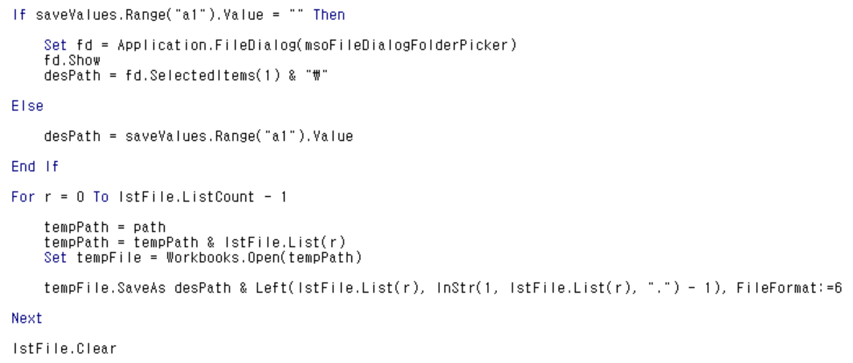


The following block of codes will be executed when a specific file is selected from the list. Previously I said that I will come back to the topic of converting files in different types of extension. The line that has FileFormat:=6 is what makes the .xls file to be converted to .csv file. Excel supports numerous types of file extension. For instance, if we take a look at the browse screen when we save a new file, it shows close to 10 different types of extension that we can select from. Each extension is designated by a different number; in the case of .csv, the designated number is 6. If I want to support another file type, I just have to ask Google for the number.



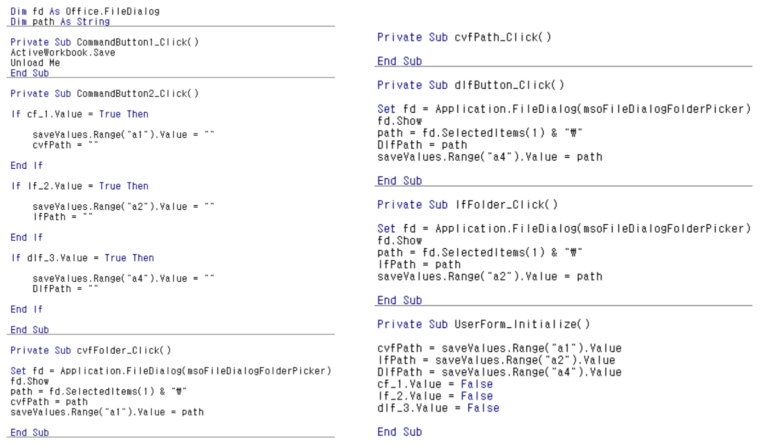
```
If saveValues.Range("a1").Value = "" Then

    Set fd = Application.FileDialog(msoFileDialogFolderPicker)
    fd.Show
    desPath = fd.SelectedItems(1) & "₩"

Else

    desPath = saveValues.Range("a1").Value

End If

tempPath = path
tempPath = tempPath & lstFile.List(lstFile.ListIndex)
Set tempFile = Workbooks.Open(tempPath)
tempFile.SaveAs desPath & Left(lstFile.List(r), InStr(1, lstFile.List(r), ".") - 1), FileFormat:=6
lstFile.RemoveItem lstFile.ListIndex
```

The below block of codes will be executed when employee doesn't specify a file to be converted. One of the differences from the below block of codes to the above block of codes is that the above removes the converted file name from the list right after each conversion process is finished whereas the below remove all of the files from the list after the whole conversion process is finished.



```
If saveValues.Range("a1").Value = "" Then

    Set fd = Application.FileDialog(msoFileDialogFolderPicker)
    fd.Show
    desPath = fd.SelectedItems(1) & "₩"

Else

    desPath = saveValues.Range("a1").Value

End If

For r = 0 To lstFile.ListCount - 1

    tempPath = path
    tempPath = tempPath & lstFile.List(r)
    Set tempFile = Workbooks.Open(tempPath)

    tempFile.SaveAs desPath & Left(lstFile.List(r), InStr(1, lstFile.List(r), ".") - 1), FileFormat:=6

Next

lstFile.Clear
```

Employee can configure the program to prompt him/her with a location of the converted files each time he/she runs the program; only thing is required to do this is to leave out the default directory path blank. To do this, I created three check boxes with a clear button. Employee can clear any of the three default directories' paths by checking the relative check boxes and double clicking the Clear button. The following code is the entire code of the default setting form.

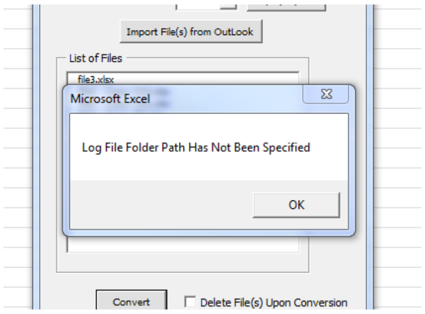


```
Dim fd As Office.FileDialog
Dim path As String

Private Sub CommandButton1_Click()
ActiveWorkbook.Save
Unload Me
End Sub

Private Sub CommandButton2_Click()

If cf_1.Value = True Then

    saveValues.Range("a1").Value = ""
    cvfPath = ""

End If

If lf_2.Value = True Then

    saveValues.Range("a2").Value = ""
    lfPath = ""

End If

If dlf_3.Value = True Then

    saveValues.Range("a4").Value = ""
    DlfPath = ""

End If

End Sub

Private Sub cvfFolder_Click()

Set fd = Application.FileDialog(msoFileDialogFolderPicker)
fd.Show
path = fd.SelectedItems(1) & "₩"
cvfPath = path
saveValues.Range("a1").Value = path

End Sub

Private Sub cvfPath_Click()

End Sub

Private Sub dlfButton_Click()

Set fd = Application.FileDialog(msoFileDialogFolderPicker)
fd.Show
path = fd.SelectedItems(1) & "₩"
DlfPath = path
saveValues.Range("a4").Value = path

End Sub

Private Sub lfFolder_Click()

Set fd = Application.FileDialog(msoFileDialogFolderPicker)
fd.Show
path = fd.SelectedItems(1) & "₩"
lfPath = path
saveValues.Range("a2").Value = path

End Sub

Private Sub UserForm_Initialize()

cvfPath = saveValues.Range("a1").Value
lfPath = saveValues.Range("a2").Value
DlfPath = saveValues.Range("a4").Value
cf_1.Value = False
lf_2.Value = False
dlf_3.Value = False

End Sub
```

Producing Log

User Case: Program produces a log after the conversion process is finished.

Third, when the Convert button is double clicked, the whole converting process will be recorded on either a new log or the default log depending on the existence of a log file. If a log file is created before and the full path is written on the spread sheet, the log will be used to record the process of conversion. If no default directory is specified, the program will not run at all instead it will prompt user to pick a default directory for a log file.



Microsoft Excel

Log File Folder Path Has Not Been Specified

The below code will be executed when the conversion process kicks in. Again, if no existing log file is found, the program will create a one for employee and put the log file into the default folder, and if program detects an existing log file then it will be used to record conversion process.

```
If saveValues.Range("a3").Value = "" Then
    tempLogPath = saveValues.Range("a2").Value & "logFile.log"
    Set f = fso.createTextFile(tempLogPath, True)
    saveValues.Range("a3").Value = tempLogPath
Else
    Set f = fso.openTextFile(saveValues.Range("a3").Value, ForAppending, True)
End If
```

One problem I had to solve before writing any further code was to specify how the text file is going to be open. Apparently, I am using ForAppending which means I am opening the text file to append new values. Other than ForAppending, there are two more options: ForWriting and ForReading. At first, I thought the function recognizes the characters, but it turned out to be that I had to set up specific numeric values for each of the category. For instance,

```
options for editting the log file
Const ForReading = 1, ForWriting = 2, ForAppending = 8

Private Sub importFiles_Click()
```

In another way of satisfying the function to be properly ran is to just write the number down, but assigning values to characters help me to know what each number represents.

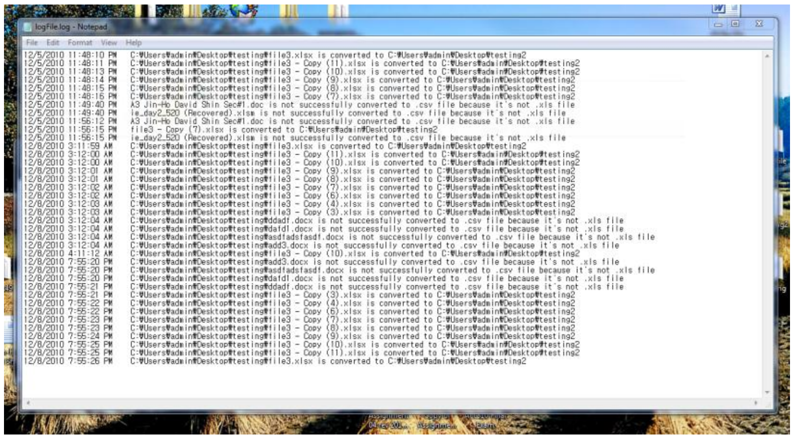
Detecting Corrupted Files

User Case: Employee is alerted when the selected files aren’t converted properly.

The hardest part of the writing this part of the programming was to figure out how I am going to detect if there is a corrupted file at all, and how I am going to alert employee. After searching Google for a while, I found out that it wasn’t hard after all. I had to just use the On Error GoTo ErrHandler again.

```
Exit Sub
ErrHandler:

If errorPass = False Then
    f.writeLine Now() & vbTab & tempPath & " is not successfully converted to .csv file because it's either a corrupted file or not a supported file extension "

    If lstFile.ListIndex <> -1 Then
        lstFile.RemoveItem lstFile.ListIndex
    End If

    errorPass = True
End If

Resume Next
```

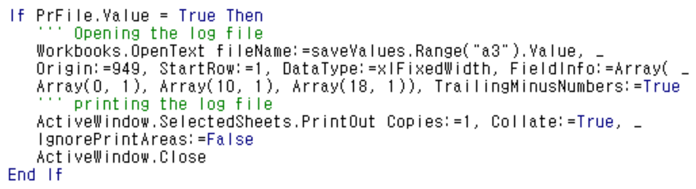
The block of code above takes care of writing a log when an error-caused file is detected. I had to surround the F.writeLine…. code with errorPass = Pass if-statement; because, without these codes, the line of the code will not work if two error-caused files are detected in a list of files that are going through a loop; let say that a corrupted file is detected while a batch of 10 files are going through a loop and there are still 5 more files to check if they are legal files. Detecting the corrupted file will pretty much kill the program unless I put resume next at the end of ErrorHandler. Putting the resume next at the end of the code will solve the problem of not checking the rest of 5 files, but what if none of the files are corrupted? Without the if-statement, the entry of the error caused file will be written 6 times on the log; therefore, I had to include the if-statement in that place to prevent the log file to be written multiple times with the same error-caused file. After the log file is written, the log file will be looked like the screen shot below.



```
12/5/2010 11:48:10 PM    C:₩Users₩admin₩Desktop₩testing₩file3.xlsx is converted to C:₩Users₩admin₩Desktop₩testing2
12/5/2010 11:48:11 PM    C:₩Users₩admin₩Desktop₩testing₩file3 - Copy (11).xlsx is converted to C:₩Users₩admin₩Desktop₩testing2
12/5/2010 11:48:13 PM    C:₩Users₩admin₩Desktop₩testing₩file3 - Copy (10).xlsx is converted to C:₩Users₩admin₩Desktop₩testing2
12/5/2010 11:48:14 PM    C:₩Users₩admin₩Desktop₩testing₩file3 - Copy (9).xlsx is converted to C:₩Users₩admin₩Desktop₩testing2
12/5/2010 11:48:15 PM    C:₩Users₩admin₩Desktop₩testing₩file3 - Copy (8).xlsx is converted to C:₩Users₩admin₩Desktop₩testing2
12/5/2010 11:48:16 PM    C:₩Users₩admin₩Desktop₩testing₩file3 - Copy (7).xlsx is converted to C:₩Users₩admin₩Desktop₩testing2
12/5/2010 11:49:40 PM    A3 Jin-Ho David Shin Sec#1.doc is not successfully converted to .csv file because it's not .xls file
12/5/2010 11:49:40 PM    ie_day2_520 (Recovered).xlsm is not successfully converted to .csv file because it's not .xls file
12/5/2010 11:56:12 PM    A3 Jin-Ho David Shin Sec#1.doc is not successfully converted to .csv file because it's not .xls file
12/5/2010 11:56:15 PM    file3 - Copy (7).xlsx is converted to C:₩Users₩admin₩Desktop₩testing2
12/5/2010 11:56:15 PM    ie_day2_520 (Recovered).xlsm is not successfully converted to .csv file because it's not .xls file
12/8/2010 3:11:59 AM     C:₩Users₩admin₩Desktop₩testing₩file3.xlsx is converted to C:₩Users₩admin₩Desktop₩testing2
12/8/2010 3:12:00 AM     C:₩Users₩admin₩Desktop₩testing₩file3 - Copy (11).xlsx is converted to C:₩Users₩admin₩Desktop₩testing2
12/8/2010 3:12:00 AM     C:₩Users₩admin₩Desktop₩testing₩file3 - Copy (10).xlsx is converted to C:₩Users₩admin₩Desktop₩testing2
12/8/2010 3:12:01 AM     C:₩Users₩admin₩Desktop₩testing₩file3 - Copy (9).xlsx is converted to C:₩Users₩admin₩Desktop₩testing2
12/8/2010 3:12:01 AM     C:₩Users₩admin₩Desktop₩testing₩file3 - Copy (8).xlsx is converted to C:₩Users₩admin₩Desktop₩testing2
12/8/2010 3:12:02 AM     C:₩Users₩admin₩Desktop₩testing₩file3 - Copy (7).xlsx is converted to C:₩Users₩admin₩Desktop₩testing2
12/8/2010 3:12:02 AM     C:₩Users₩admin₩Desktop₩testing₩file3 - Copy (6).xlsx is converted to C:₩Users₩admin₩Desktop₩testing2
12/8/2010 3:12:03 AM     C:₩Users₩admin₩Desktop₩testing₩file3 - Copy (4).xlsx is converted to C:₩Users₩admin₩Desktop₩testing2
12/8/2010 3:12:03 AM     C:₩Users₩admin₩Desktop₩testing₩file3 - Copy (3).xlsx is converted to C:₩Users₩admin₩Desktop₩testing2
12/8/2010 3:12:04 AM     C:₩Users₩admin₩Desktop₩testing₩ddadf.docx is not successfully converted to .csv file because it's not .xls file
12/8/2010 3:12:04 AM     C:₩Users₩admin₩Desktop₩testing₩dafdl.docx is not successfully converted to .csv file because it's not .xls file
12/8/2010 3:12:04 AM     C:₩Users₩admin₩Desktop₩testing₩asdfadsfasdf.docx is not successfully converted to .csv file because it's not .xls file
12/8/2010 3:12:04 AM     C:₩Users₩admin₩Desktop₩testing₩add3.docx is not successfully converted to .csv file because it's not .xls file
12/8/2010 4:11:12 AM     C:₩Users₩admin₩Desktop₩testing₩file3 - Copy (10).xlsx is converted to C:₩Users₩admin₩Desktop₩testing2
12/8/2010 7:55:20 PM     C:₩Users₩admin₩Desktop₩testing₩add3.docx is not successfully converted to .csv file because it's not .xls file
12/8/2010 7:55:20 PM     C:₩Users₩admin₩Desktop₩testing₩asdfadsfasdf.docx is not successfully converted to .csv file because it's not .xls file
12/8/2010 7:55:20 PM     C:₩Users₩admin₩Desktop₩testing₩dafdl.docx is not successfully converted to .csv file because it's not .xls file
12/8/2010 7:55:21 PM     C:₩Users₩admin₩Desktop₩testing₩ddadf.docx is not successfully converted to .csv file because it's not .xls file
12/8/2010 7:55:21 PM     C:₩Users₩admin₩Desktop₩testing₩file3 - Copy (3).xlsx is converted to C:₩Users₩admin₩Desktop₩testing2
12/8/2010 7:55:22 PM     C:₩Users₩admin₩Desktop₩testing₩file3 - Copy (4).xlsx is converted to C:₩Users₩admin₩Desktop₩testing2
12/8/2010 7:55:22 PM     C:₩Users₩admin₩Desktop₩testing₩file3 - Copy (6).xlsx is converted to C:₩Users₩admin₩Desktop₩testing2
12/8/2010 7:55:23 PM     C:₩Users₩admin₩Desktop₩testing₩file3 - Copy (7).xlsx is converted to C:₩Users₩admin₩Desktop₩testing2
12/8/2010 7:55:23 PM     C:₩Users₩admin₩Desktop₩testing₩file3 - Copy (8).xlsx is converted to C:₩Users₩admin₩Desktop₩testing2
12/8/2010 7:55:24 PM     C:₩Users₩admin₩Desktop₩testing₩file3 - Copy (9).xlsx is converted to C:₩Users₩admin₩Desktop₩testing2
12/8/2010 7:55:25 PM     C:₩Users₩admin₩Desktop₩testing₩file3 - Copy (10).xlsx is converted to C:₩Users₩admin₩Desktop₩testing2
12/8/2010 7:55:25 PM     C:₩Users₩admin₩Desktop₩testing₩file3 - Copy (11).xlsx is converted to C:₩Users₩admin₩Desktop₩testing2
12/8/2010 7:55:26 PM     C:₩Users₩admin₩Desktop₩testing₩file3.xlsx is converted to C:₩Users₩admin₩Desktop₩testing2
```

It's very hard to see, but if we look at the log closely, we can see that there are multiple log entries that say "is not successfully converted to .csv file because it's either a corrupted file or not a supported file extension." I could if I want to prompt a warning message to employee every time file is not properly converted with MsgBox function, but I felt that it will be a bothersome to have that warning message pop up multiple times, so I decided to write it down on the log to have employee look at it after the conversion process is finished. Employee will not have hard time finding which log entries apply to the one that is just finished because I wrote the program to add a time stamp each time file is converted.

Fourth, sometimes employee feel more comfortable to see the log file in a hard copy paper than on the screen; therefore, I add a functionality that employee can choose to print the log file right after the conversion process is finished. Employee simply needs to check ☐ Print Log File box. The following block of code is how the functionality works in codes.



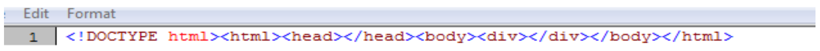
```
If PrFile.Value = True Then
   ''' Opening the log file
   Workbooks.OpenText fileName:=saveValues.Range("a3").Value, _
   Origin:=949, StartRow:=1, DataType:=xlFixedWidth, FieldInfo:=Array( _
   Array(0, 1), Array(10, 1), Array(18, 1)), TrailingMinusNumbers:=True
   ''' printing the log file
   ActiveWindow.SelectedSheets.PrintOut Copies:=1, Collate:=True, _
   IgnorePrintAreas:=False
   ActiveWindow.Close
End If
```

One thing that employee needs to make sure before printing the log file is that the default printer is valid. We can simply set the default printer by printing a paper; this way will leave the working printer in the priority.

Download files from Outlook (Internet)

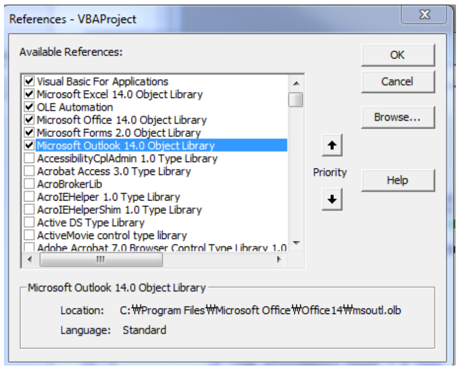
User Case: Program downloads attachments of emails and lists the files in the GUID and put them in the default folder.

This was the hardest part of the project. I couldn't use Dr. Gove's class module because HTML on email client did not work in the way how other webpages work. Let me explain it what it means; it took me a long time, but I was able to log into my Gmail account by learning the html code. After logged in, it was dead end because this is what I got:



```
<!DOCTYPE html><html><head></head><body><div></div></body></html>
```

I found very interesting that viewing source page with Explorer didn't get me anything like the screen shot above, but with FirefFox, I was able to get it. Unfortunately, Excel only supports Explorer, so I had to figure out another approach to solve the problem. After long hours of researching, Outlook object came to my attention. At first, I thought Outlook object only works with Outlook application, but I found out that I could manipulate Outlook object by adding this reference to Excel.



References - VBAProject

Available References:

- [✓] Visual Basic For Applications
- [✓] Microsoft Excel 14.0 Object Library
- [✓] OLE Automation
- [✓] Microsoft Office 14.0 Object Library
- [✓] Microsoft Forms 2.0 Object Library
- [✓] Microsoft Outlook 14.0 Object Library
- [ ] AccessibilityCplAdmin 1.0 Type Library
- [ ] Acrobat Access 3.0 Type Library
- [ ] AcroBrokerLib
- [ ] AcroIEHelper 1.0 Type Library
- [ ] AcroIEHelperShim 1.0 Type Library
- [ ] Active DS Type Library
- [ ] ActiveMovie control type library
- [ ] Adobe Acrobat 7.0 Browser Control Type Library 1.0

Microsoft Outlook 14.0 Object Library

Location: C:₩Program Files₩Microsoft Office₩Office14₩msoutl.olb

Language: Standard

This reference could be found by going Tools → Reference → scroll down till you see the highlighted element→ check the check box.

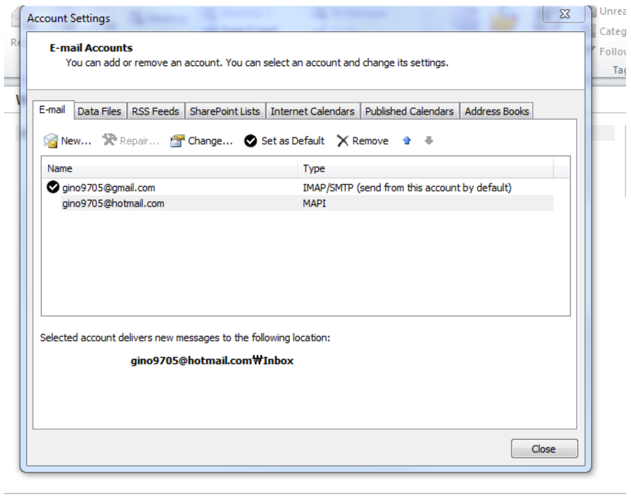
Finding out about Outlook object took me a long time, but learning how to manipulate took me longer mainly because most of the tutorials I found were referring to Outlook application not Excel making me to believe that Excel doesn't support this object. As a matter of fact, I had to dissect the object into pieces to learn the usage. This is what I did to see what properties and methods are available for each element.

```
Dim oLookObject As Outlook.Application
Dim myNameSpace As Outlook.Namespace
Dim inbox As Outlook.Folder
Dim iTems As Outlook.iTems
Dim iTem As Outlook.MailItem
Dim atmt As Outlook.Attachment
```

The below block of codes are what I used to get email out of my Gmail. The code may seem very little but to run this code properly I had learn basic functionalities of Outlook application.

```
Set oLookObject = CreateObject("outlook.application")
Set myNameSpace = oLookObject.GetNamespace("MAPI")
'Set inbox = myNameSpace.GetDefaultFolder(olFolderInbox)
Set inbox = myNameSpace.PickFolder
'Set inbox = myNameSpace.GetFolderFromID("00000000DDE801156753A549B0CB9FECE005D18C82800000")
Set iTems = inbox.iTems

For Each iTem In iTems

    ' the first three letters need to be "ABC"
    If Left(iTem.Subject, 3) = "abc" Then
        'get only unread emails
        If iTem.UnRead = True Then
            If iTem.Attachments.Count > 0 Then
                For Each atmt In iTem.Attachments

                    'fileName = Now()
                    'fileName = Replace(fileName, "/", "_")
                    'fileName = Replace(fileName, ":", "_")
                    fileName = saveValues.Range("a4") & atmt.fileName

                    ReDim Preserve orgArray(r)
                    orgArray(r) = atmt.fileName

                    atmt.SaveAsFile (fileName)

                    path = saveValues.Range("a4")

                    iTem.UnRead = False
                    r = r + 1
                Next
            End If
        End If
    End If
Next
```

The line above .GetNameSpace("MAPI") gives me a way to manipulate Outlook object, and GetDefaultFolder(olFolderInbox) supposed to get me the default folder of Outlook account. I learned that Outlook application allow users to add numerous email accounts so that users don't have to navigate one email account to another but have all of them in one controlling place. To run the code, employee needs to add an account in Outlook application, and most of the employees working OIT BYU use outlook so it will be a cake walk for them to do that. Setting up a default account was the trickiest of all. Supposedly, I could set up a default account by going to account setting like the screen shot below, but I kept getting outlook data file as a default email address which is a back-up storage for other email accounts.



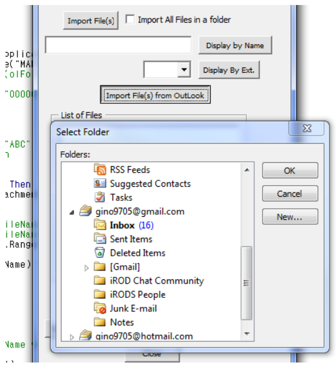
You can add or remove an account. You can select an account and change its settings.

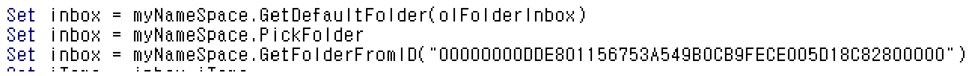
Name	Type
✓ gino9705@gmail.com	IMAP/SMTP (send from this account by default)
gino9705@hotmail.com	MAPI

Selected account delivers new messages to the following location:

gino9705@hotmail.com₩Inbox

I had to do lots of things before I found out that I could actually have employee pick email account each time he/she tries to load attached files in the interface; second one of the list below will do that.





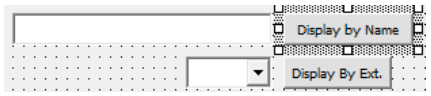
```
Set inbox = myNameSpace.GetDefaultFolder(olFolderInbox)
Set inbox = myNameSpace.PickFolder
Set inbox = myNameSpace.GetFolderFromID("00000000DDE801156753A549B0CB9FECE005D018C82800000")
```

I don’t know if it’s effective but I could technically get the id of a specific id of an account and use that to retrieve attached files. I could give several barriers to weed out emails to get the certain emails; for instance, I could make the program to get only unread message with a certain heading. In the example of my code, I am requiring the email to be unread and have an ABC letter starting in the heading as barriers to retrieve data. After downloading all of the attachments in the default directory (if no default directory is being specified, employee will be warned and the program will not urn), the files will be listed in the interface in the same manner of the ones I explained before. Sometimes, Outlooks takes a while to recognize new coming emails; it’s recommended to open Outlook application before running the program.

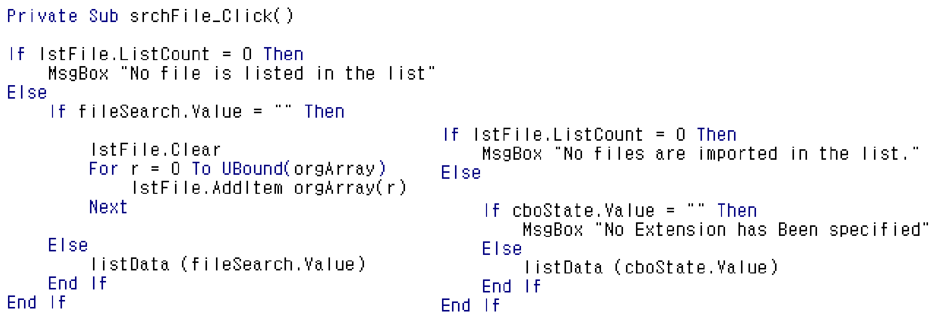
Search File

User Case: Employee searches a specific file from the list of the uploaded files by typing in the input box.

Adding this functionality was the easiest of all mainly because I had to do similar thing on my homework project. I made possible for employee to search a specific file either by letters or by extension like the following screen shot.



Before employee is able to search for a specific file, I made an array of all files listed in the interface when employee double clicks Import File(s) button. I used the following block of code to do the search query by letter(s) or by an extension (.xls extension is only extension in this program because that is what my employer asked me to do).



```
Private Sub srchFile_Click()

If lstFile.ListCount = 0 Then
    MsgBox "No file is listed in the list"
Else
    If fileSearch.Value = "" Then

        lstFile.Clear
        For r = 0 To UBound(orgArray)
            lstFile.AddItem orgArray(r)
        Next

    Else
        listData (fileSearch.Value)
    End If
End If
```

```
If lstFile.ListCount = 0 Then
    MsgBox "No files are imported in the list."
Else

    If cboState.Value = "" Then
        MsgBox "No Extension has Been specified"
    Else
        listData (cboState.Value)
    End If
End If
```

Employee will be able to close the program without an error by double clicking the Close button.