Aaron Fisher
4/13/11

VBA Semester Project: Email to Spreadsheet Parser

Executive Summary

For my semester project I began by addressing a business problem in the place where I work, Seven Peaks Water Park. As an employee in the IT department as well as the Point of Sale Specialist, I receive a lot of emails from automated devices that fill up my inbox, as well as emails from various managers requesting timely information that may go unnoticed if my inbox fills up quickly with less important messages. One particular email that I receive several times a day is generated from our website with customer information when a patron requests information about “group events” (i.e. a family party they would like to hold at a discounted rate).

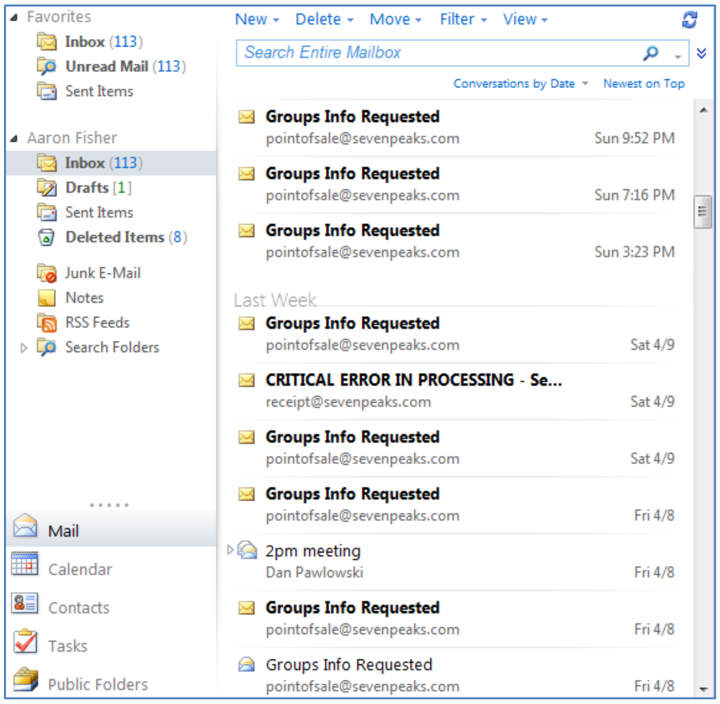
These emails have been collecting in my inbox during the off season and will eventually need to be passed on to an employee responsible for contacting each one. The preferred method of distributing contact information at this business is spreadsheet form, so I created an outlook marco in VBA that iterates through my outlook inbox and finds automated message of this type, then parses the contents of the email and extracts the important information using regular expressions and appends it to an excel spreadsheet so that no person is left un-contacted and the process of contacting each person is greatly facilitated by the fact that its presented neatly in a spread sheet and no one has to transfer it from emails to user-friendly form.

*In addition to this business solution, I challenged myself a bit further by creating an excel-based computer game done entirely in VBA. I named this game Rocket Town; the objective is to navigate a rocket around obstacles that are randomly placed in your path. This application presented many unforeseen challenges and proved to be an excellent learning exercise.

Implementation

This section is a more detailed explanation of the business problem and the solution – but still on a high-level view. For low-level detail of how the solution works on the programming level, see the “Write-up Detail” section.

The problem:



Groups Info Requested
pointofsale@sevenpeaks.com Sun 9:52 PM

Groups Info Requested
pointofsale@sevenpeaks.com Sun 7:16 PM

Groups Info Requested
pointofsale@sevenpeaks.com Sun 3:23 PM

Last Week

Groups Info Requested
pointofsale@sevenpeaks.com Sat 4/9

CRITICAL ERROR IN PROCESSING - Se...
receipt@sevenpeaks.com Sat 4/9

Groups Info Requested
pointofsale@sevenpeaks.com Sat 4/9

Groups Info Requested
pointofsale@sevenpeaks.com Fri 4/8

2pm meeting
Dan Pawlowski Fri 4/8

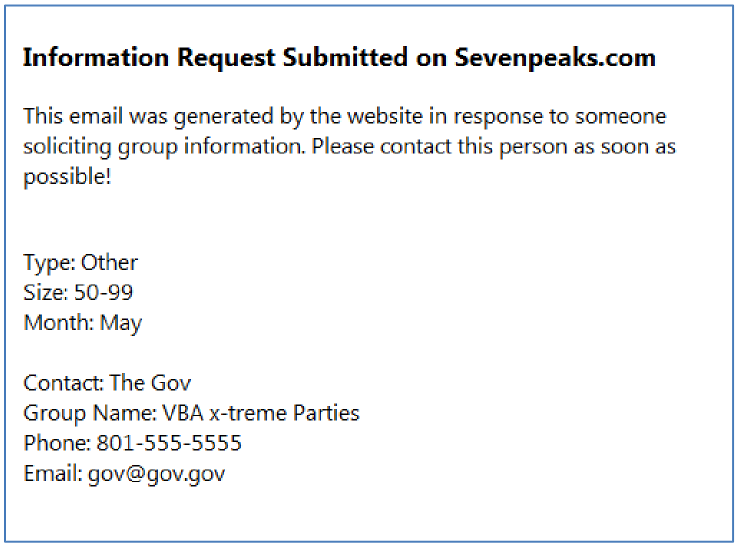
Groups Info Requested
pointofsale@sevenpeaks.com Fri 4/8

Groups Info Requested
pointofsale@sevenpeaks.com Fri 4/8

This figure is a screen shot of my outlook inbox. This figure demonstrates the problem visually, which is that eight out of ten of the visible emails at this particular time are “Groups Info Requested” emails, which are automatically generated from the website. You can see that there is also a 2pm meeting item which contains important information about a meeting scheduled that same day. By receiving so many auto-generated request items, I lose the valuable ability to either glance at my inbox to see important and timely information, or to monitor incoming emails through desktop notifications. It’s essentially the same problem as SPAM, only all of the information is important and represents a possible revenue source for the business, so it must be processed. There is an easy solution to the email crowding out important messages, which is to create a filter and have outlook sort the messages as they are received, which I may still implement in the future, but this is not a complete solution.

If the email where categorized so that it didn’t interfere with my regular emailing activities, the information in the email still needs to reach the right person. (I should mention that I receive the emails only because I built the webpage that generates the emails and at the time, we did not have an employee is this position and will not have one until the park is open for business again.)

Here is a sample email auto generated by the website to put this problem into context.



Information Request Submitted on Sevenpeaks.com

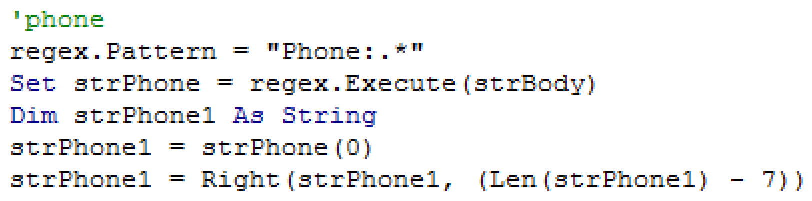
This email was generated by the website in response to someone soliciting group information. Please contact this person as soon as possible!

Type: Other
Size: 50-99
Month: May

Contact: The Gov
Group Name: VBA x-treme Parties
Phone: 801-555-5555
Email: gov@gov.gov

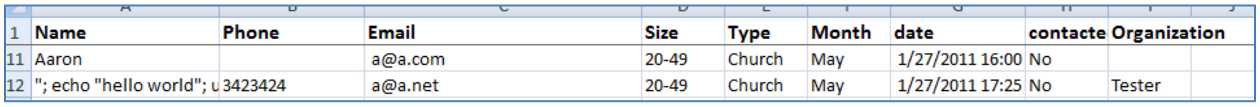
As you can see, the email contains essential information for contacting the individual, and dozens of these emails are generated each week. To forward these emails on to the person or persons responsible for contacting each individual means managing hundreds of individual files, and iterating through each email as the employee contacts them, and then recording related notes with in another file. Obviously, this information could be presented in a spreadsheet where it could be handled as one document. Looking at the sample email you can see that email template was structured cleverly (by me) so that information could later be extracted out of it.

I decided that the implementation of regular expressions would be an appropriate way to extract the data so it could then be placed into an excel spreadsheet. Adding a regular expressions library, I was able to extract the information with some very simple regular expressions shown in the next figure.



```
'phone
regex.Pattern = "Phone:.*"
Set strPhone = regex.Execute(strBody)
Dim strPhone1 As String
strPhone1 = strPhone(0)
strPhone1 = Right(strPhone1, (Len(strPhone1) - 7))
```

Using the regular expression pattern I was able to extract all of the important information from each email and store it in a variable within the macro. I then used VBA from within Outlook to instantiate excel, open the spreadsheet file from network storage, and write the email information into fields in the excel spreadsheet with additional information for the employee responsible for contacting each person.



	Name	Phone	Email	Size	Type	Month	date	contacte	Organization	
11	Aaron		a@a.com	20-49	Church	May	1/27/2011 16:00	No		
12	"; echo "hello world"; u	3423424	a@a.net	20-49	Church	May	1/27/2011 17:25	No	Tester	

This is what the actual spreadsheet looks like. To protect customers' personal contact information I only displayed some sample test data that came from the website, was pulled from Outlook, and written to the spreadsheet.

I also used the VBA script to mark each message as read and then move it to a different folder where I can manually retrieve it in the event that some mishap occurs in the script.

That's the context of the solution. I'll explain in more detail the actual programming elements of the script in the following sections.

Discussion of Learning and Conceptual Difficulties

This project was fairly simple. For that reason I believe this was an excellent and very appropriate application of VBA. I did run into several difficulties while developing this script, however, because they were not conceptually very difficult, I chose to do an additional project to supplement this one.

In the game I developed I ran into a whole new world of problems and programming concerns that I had never dealt with in my business orientated programming education. Those challenges are documented separately.

In this application, the first major challenge I ran into was dealing with the outlook application schema which is different from the excel layout that I've used throughout the semester. Conceptually, I had to move from using workbooks, sheets, and cells to folders and items. This was initially confusing but I was able to find sample code to help me navigate to the inbox and iterate through the items contained therein.

The next problem I ran into was the task of moving data from the Outlook application to Excel. I knew I wanted the information to end up in a spreadsheet, but I wasn't sure how I could physically move data from the macro I started in Outlook to a spreadsheet. I found that from within any of the Microsoft Office applications with VBA I can instantiate another Office application and call its methods from within my working macro. I discovered that you can import many different libraries to create new objects in macro and interact with them.

The next problem I ran into was scraping the actual email item to identify the important data. When I wrote the webpage to send the email, I thought I would probably want to scrape the data programmatically, or at least put the information in a format where it was easy to find the with the human eye. I put each important item on a separate line with a label; I was able to identify it with some pretty simple regular expressions. To use the regular expressions I had to import an additional library. I also ran into some trouble because I found that the code to execute a regular expression pattern returns an array, each matching item being an item in the array –which makes sense, but because I was only expecting one match I expected to receive a string. So when I figured out what was going on I had to manipulate the variables by saving the 0^{th} item of each array as a string, and then storing each string into another array of scraped data which was my final return from the function.

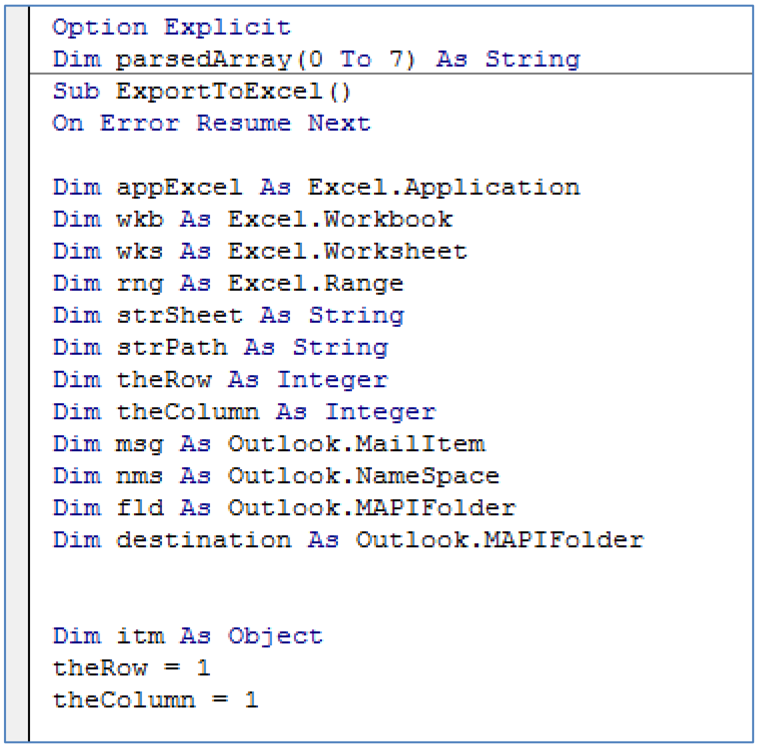
After the email is scraped, the next problem I ran into was inputting the data into spreadsheet form. Obviously it would be less than ideal to create a new spreadsheet every time, so I decided to reference a spreadsheet which I use every time and find the first blank row to start writing.

The final detail was marking the messages as read and moving them to a different folder so as to clean up my in box. I was able to accomplish this by investigating the object's properties, where I found methods to mark as read and move.

Write Up Detail

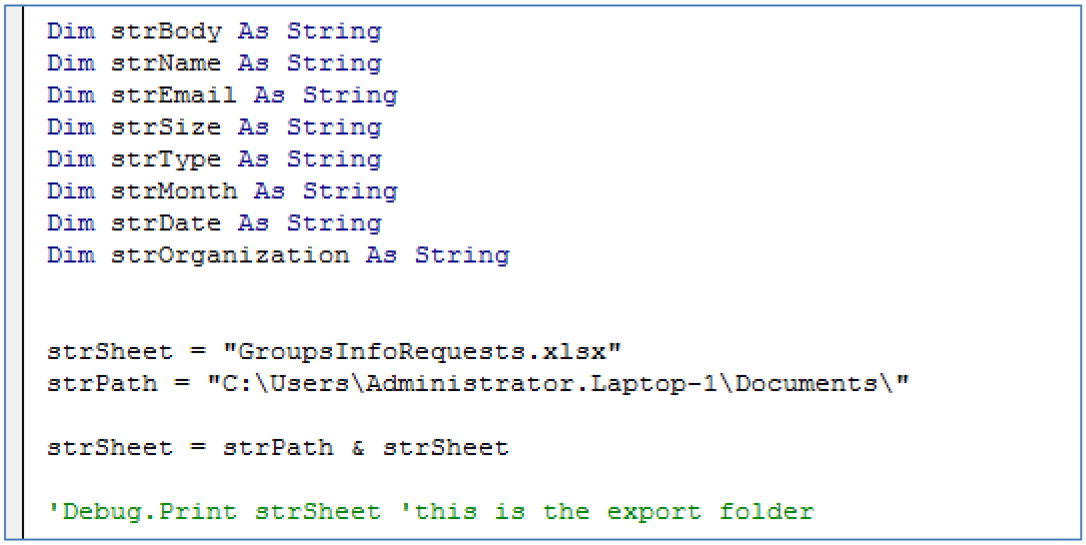
I believe the simplest way to describe my script on the programming level is to paste in the code in snippets and explain them.

This is the start of the script where I just declare the variables I used for both the excel and outlook objects.



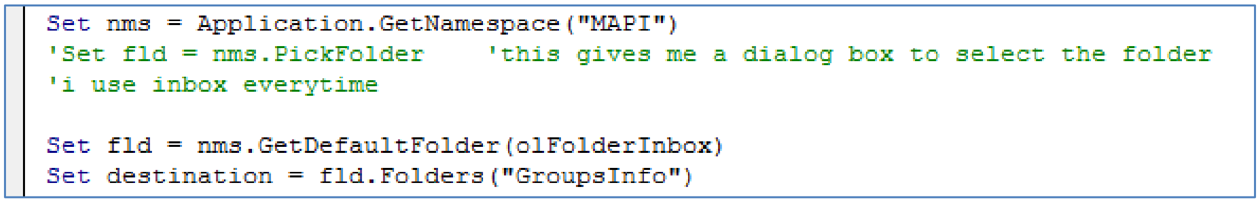
```
Option Explicit
Dim parsedArray(0 To 7) As String
Sub ExportToExcel()
On Error Resume Next

Dim appExcel As Excel.Application
Dim wkb As Excel.Workbook
Dim wks As Excel.Worksheet
Dim rng As Excel.Range
Dim strSheet As String
Dim strPath As String
Dim theRow As Integer
Dim theColumn As Integer
Dim msg As Outlook.MailItem
Dim nms As Outlook.NameSpace
Dim fld As Outlook.MAPIFolder
Dim destination As Outlook.MAPIFolder


Dim itm As Object
theRow = 1
theColumn = 1
```

In the next snippet of code is where I declared variables for content manipulation. I have variables for each of the fields that I need to pull from the email, as well as a file path.



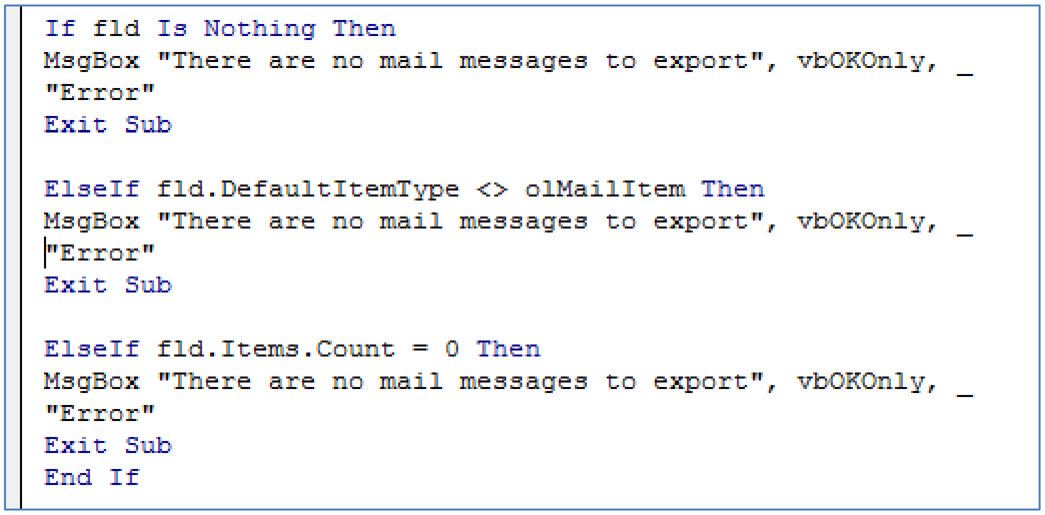
```
Dim strBody As String
Dim strName As String
Dim strEmail As String
Dim strSize As String
Dim strType As String
Dim strMonth As String
Dim strDate As String
Dim strOrganization As String


strSheet = "GroupsInfoRequests.xlsx"
strPath = "C:\Users\Administrator.Laptop-1\Documents\"

strSheet = strPath & strSheet

'Debug.Print strSheet 'this is the export folder
```

After declaring my variables, I set the outlook variables. Here you can see I could specify a folder to use as I iterate through to check emails. I specified the inbox folder because I'll always use that and I want the script to be fully automated. Before I made that decision I had a file picker allow the user to select which folder to use. I also set a destination folder to move the emails to after they've been recorded.



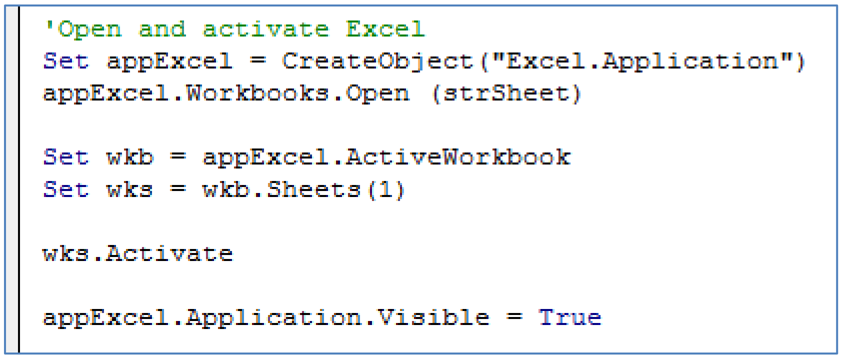
```
Set nms = Application.GetNamespace("MAPI")
'Set fld = nms.PickFolder    'this gives me a dialog box to select the folder
'i use inbox everytime

Set fld = nms.GetDefaultFolder(olFolderInbox)
Set destination = fld.Folders("GroupsInfo")
```

Here I've implemented some error checking. The idea here is to report back if no email items are found in the specified folder.



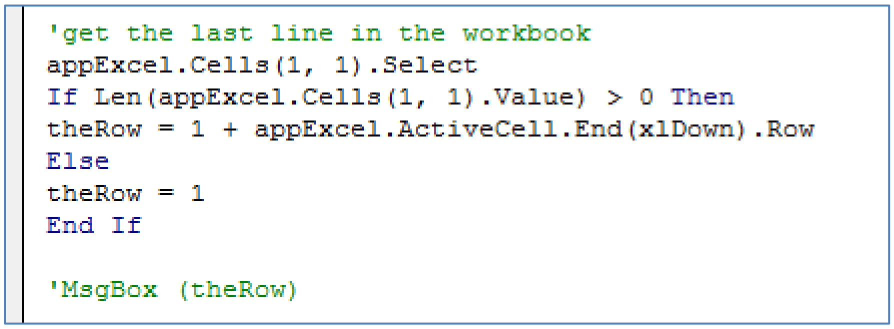
```
If fld Is Nothing Then
MsgBox "There are no mail messages to export", vbOKOnly, _
"Error"
Exit Sub

ElseIf fld.DefaultItemType <> olMailItem Then
MsgBox "There are no mail messages to export", vbOKOnly, _
"Error"
Exit Sub

ElseIf fld.Items.Count = 0 Then
MsgBox "There are no mail messages to export", vbOKOnly, _
"Error"
Exit Sub
End If
```

Once I found items in my folder I found the next step was to instantiate excel. The following snippet is how I created an instance of excel from within outlook VBA.



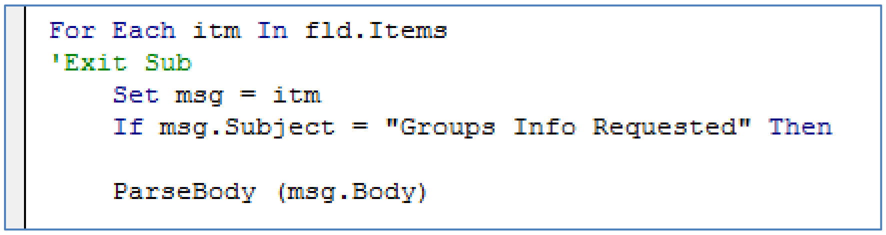
```
'Open and activate Excel
Set appExcel = CreateObject("Excel.Application")
appExcel.Workbooks.Open (strSheet)

Set wkb = appExcel.ActiveWorkbook
Set wks = wkb.Sheets(1)

wks.Activate

appExcel.Application.Visible = True
```

Because I want to append to the same spreadsheet, I have to find the last line of the file I specified and save the row number so that I can append from that point.



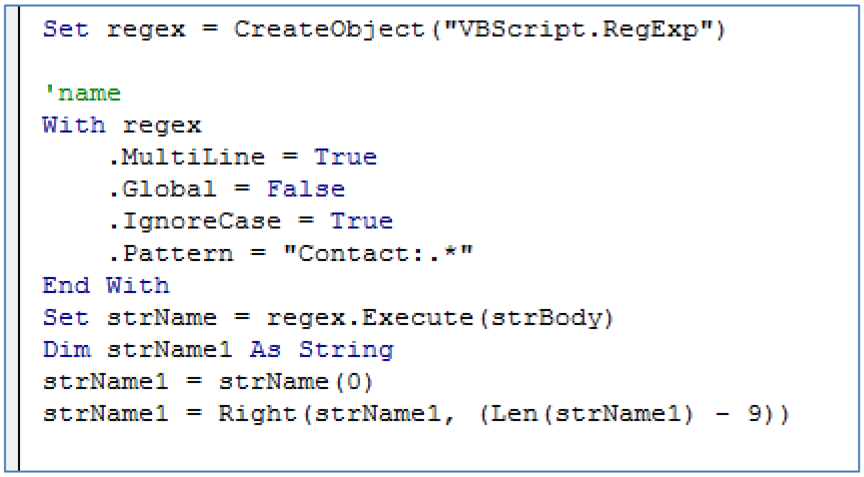
```
'get the last line in the workbook
appExcel.Cells(1, 1).Select
If Len(appExcel.Cells(1, 1).Value) > 0 Then
theRow = 1 + appExcel.ActiveCell.End(xlDown).Row
Else
theRow = 1
End If

'MsgBox (theRow)
```

In this snippet I actually iterate through each email object, check if its subject line matches the auto-generated message I'm looking for, and if it does, I run my own function "ParseBody." That function passes in the email message's body and within the function I use regular expression and return an array of strings containing the information I'm looking for.



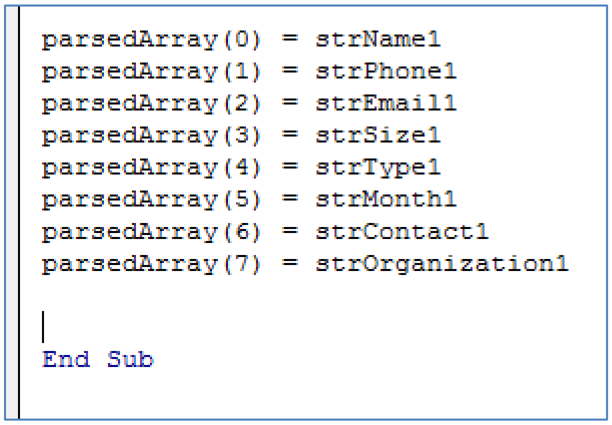
```
For Each itm In fld.Items
'Exit Sub
    Set msg = itm
    If msg.Subject = "Groups Info Requested" Then

    ParseBody (msg.Body)
```

The next snippet shows the regular expression object I created and properties I set, including the regular expression pattern to find the "Contact:" line in the email. When I execute the expression with strName = regex.Execute (strBody) I receive an array in return. As I mentioned previously, I was expecting just a string but got an array, so I declared another string and then saved the 0th indexed value of the array to that new string.



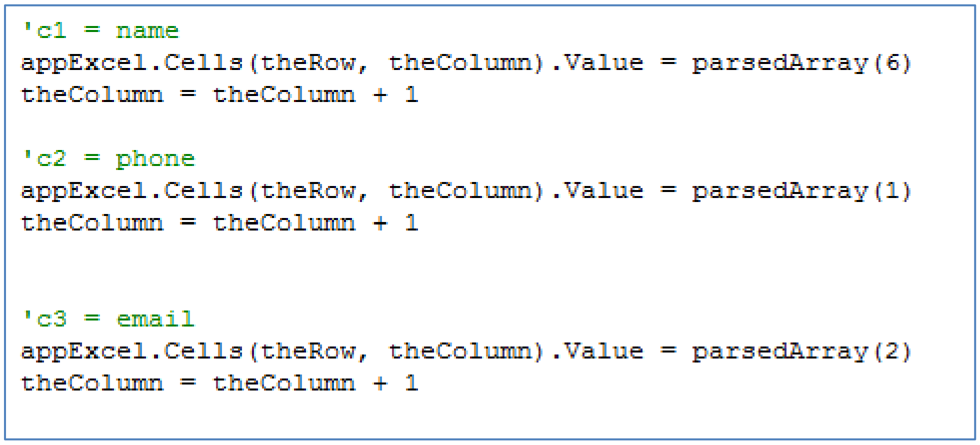
```
Set regex = CreateObject("VBScript.RegExp")

'name
With regex
    .MultiLine = True
    .Global = False
    .IgnoreCase = True
    .Pattern = "Contact:.*"
End With
Set strName = regex.Execute(strBody)
Dim strName1 As String
strName1 = strName(0)
strName1 = Right(strName1, (Len(strName1) - 9))
```

I repeated this exact process for each of the values I needed to extract from the email. Then I saved each value to an array. I returned that array from the function.



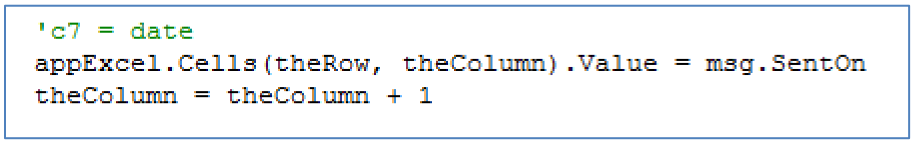
```
parsedArray(0) = strName1
parsedArray(1) = strPhone1
parsedArray(2) = strEmail1
parsedArray(3) = strSize1
parsedArray(4) = strType1
parsedArray(5) = strMonth1
parsedArray(6) = strContact1
parsedArray(7) = strOrganization1


End Sub
```

After the function is run and I have populated the “parsedArray” array, I wrote those values to the spreadsheet in the appropriate column and row. The values for the row and column are tracked in incremented to keep the cursor moving.



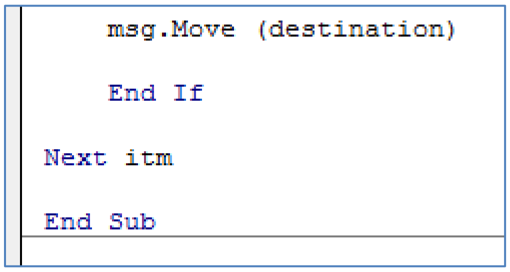
```
'c1 = name
appExcel.Cells(theRow, theColumn).Value = parsedArray(6)
theColumn = theColumn + 1

'c2 = phone
appExcel.Cells(theRow, theColumn).Value = parsedArray(1)
theColumn = theColumn + 1


'c3 = email
appExcel.Cells(theRow, theColumn).Value = parsedArray(2)
theColumn = theColumn + 1
```

In the next snippet I used the email's timestamp to record a date on the spreadsheet. I also wrote to the spreadsheet an additional hard coded value of "No" to signify that the person has not yet been contacted.



```
'c7 = date
appExcel.Cells(theRow, theColumn).Value = msg.SentOn
theColumn = theColumn + 1
```

Finally, after the message has been recorded to the spreadsheet, I move the message from the inbox to another folder so that it doesn't clutter my inbox.



```
        msg.Move (destination)

        End If

Next itm

End Sub
```

VBA Game information

Conceptual Challenges

The game presented many more challenges than the business solution. In a nut shell, I made good use of named ranges offset each range by one to simulate motion. After each cycle I check the named ranges to see if any overlap. If the rocket overlaps with an object, there has been a collision and the game is over. When the object reaches the center of the screen, the next object is generated. When the object reaches the end of the screen, it is destroy.

Conclusion

This VBA script solved a business problem and has saved many hours of labor by automating a simple process. I believe this is an appropriate application of VBA and a good business solution. I'm pleased with the outcome.

The game I created was much more challenging and open my mind to new issues I had never considered in my previous programming projects. I consider the game to be somewhat of a gateway into a new type of programming that I hope to explorer in my education.