PhUSE EU Connect 2018

Paper SM04

To VBA or not to VBA? That is the question.

Piotr Karasiewicz, Quanticate International Ltd, Warsaw, Poland

ABSTRACT

More experienced programmers – especially if they are fluent in multiple programming languages – will face the dilemma of choosing the best method to achieve a given goal. In this paper we will discuss automated delivery of company emails. There are many situations which require sending tens or hundreds of messages to individuals and people tend to use creative methods to avoid the manual distribution. Some of the possible approaches involve using SAS®, Microsoft Visual Basic for Applications (VBA) or even a combination of both. First, we will present a solution based on EMAIL (SMTP) access method within SAS FILENAME statement and then we will compare this methodology with the technique using Microsoft Visual Basic. This section will also be an opportunity to give introduction to the object oriented type of programming. Further discussion will be on merits and limitations of both routines and finally the paper will consider integration of both processes.

INTRODUCTION

This paper helps in a specific task – sending emails in bulk to a pre-specified group of recipients. It is assumed that there may be user-specific attachments and that subject/text of the email may require some level of customization depending on the addressee.

Further in the paper the list of recipients will be referred to as “email list.xlsx” or “email_list.sas7bdat” depending on whether VBA or SAS is used.

The email list is assumed to have the following columns:

- Name and Surname of recipient
- Email of recipient
- Subject of email
- Custom text of email (blank if standard message is expected)
- Path and name of attachment (blank if no attachment)

SENDING EMAILS USING SAS

SET-UP

SAS allows for three types of email protocols (this paper focuses on the first two):

- MAPI - Messaging Application Program Interface – default – it uses default email client installed on PC, e.g. Outlook or Mozilla Thunderbird.
- SMTP - Simple Mail Transfer Protocol – this method bypasses windows Client and connects to email server directly.
- VIM - Vendor Independent Mail - VIM can be used in combination with Lotus Notes or cc:Mail.

SAS option statements for MAPI & SMTP respectively:

```
OPTIONS emailsys=MAPI emailid="Piotr.Karasiewicz";
OPTIONS emailsys=SMTP emailhost="smtp.server.local" emailport=25
  emailid="Piotr.karasiewicz@quanticate.com";
```

In the case of MAPI the emailid option is usually the local-part of the email address up to and excluding “@”. It is case sensitive so it should exactly reflect the name of account in the email client, otherwise it will depend on whether the receiving host is setup to deliver in a case insensitive manner or not.

In some cases SMTP may be better solution as it doesn’t require any email client to be installed. Depending on the settings of the operating system it may be required to submit a password. In such cases the EMAILPW option can be used. It should be noted that it is highly improper to leave any password in SAS codes/logs. If for some reason this cannot be avoided then it is strongly recommended to use masked passwords. SAS offers PROC PWENCODE for that:

```
PROC PWENCODE in="sample password"; RUN;
SAS LOG: {SAS002}7A909E5721536E30210C4DA9255E328D518ABCB911D016EE
```

SAS options:

```
OPTIONS emailsys=SMTP emailhost="smtp.server.local" emailport=25
emailid="Piotr.karasiewicz@quanticate.com"
emailpw="{SAS002}7A909E5721536E30210C4DA9255E328D518ABCB911D016EE";
```

SENDING A SINGLE EMAIL

After email set-up procedure is completed delivery of single email is very straightforward.

```
FILENAME mailbox EMAIL to=("name1.surname1@quanticate.com")
                       cc=("name2.surname2@quanticate.com")
                       subject="PHUSE SM04"
                       attach=("path1\filename1");
DATA _null_;
   FILE mailbox;
   PUT 'Hi Piotr,';
   PUT 'Files for your analysis are attached. Enjoy!';
   PUT 'Regards,';
run;
FILENAME mailbox clear;
```

In the next section a more practical scenario will be discussed – sending multiple emails based on information from datasets.

BULK DISTRIBUTION OF EMAILS

The example above is easy but is not resolving the key objective of this paper – sending bulk emails. One of available options is looping through multiple FILENAME statements combined with the text of the email captured through dataset PUT statement. This paper focuses on more flexible solution using so called “directives”. They have the following form: !em_xxx!, where xxx is one of the following: newmsg, to, subject, cc, bcc, attach, importance, replyto, abort & send. The SAS code below illustrates usage of directives.

```
FILENAME mailbox EMAIL;
DATA _null_;
  SET email_list;
  FILE mailbox;
  PUT "!em_newmsg!";
  PUT "!em_to! " email;
  PUT "!em_subject!" subject;
  ** Output the main body of the email, for this no directive is needed **;
  IF custom_text ne '' THEN
    PUT custom_text;
  ELSE PUT "DEFAULT EMAIL TEXT";
  IF attachment ne '' THEN
    PUT "!em_attach!" attachment;
  PUT "!em_send!";
RUN;
FILENAME mailbox clear;
```

Each email is captured in the following way in the SAS log:

```
NOTE: The file MAILBOX is:
      E-Mail Access Device

Message sent
      To:          'piotr.karasiewicz@quanticate.com'
      Cc:
      Bcc:
      Subject:     Subject 1
      Attachments: "C:\Users\userid\Desktop\test.txt"
```

There are two problems with this code though. Firstly, the addressee captured in the last row will receive their email twice. This is due to the fact that an email is sent when `!em_send!` directive is explicitly used or `RUN;` is reached. An amended version of the code is included below with the additions highlighted in bold. The last line of the code prevents an email from being sent after the `RUN;` for the last record of dataset is executed.

```
FILENAME mailbox EMAIL;
DATA _null_;
  SET email_list END=EOF;
  FILE mailbox;
  […]
  PUT "!em_send!";
  IF EOF THEN "!em_abort!";
RUN;
FILENAME mailbox clear;
```

The second problem may show up if the EMAILSYS=SMTP option is used. It is quite likely that most of the messages will be thrown into the recipient’s spam folder, rather than their inbox. One of the events which triggers spam algorithms is when many emails are received from the same email address in a very short timeframe. During

testing using a dataset email_list.sas7bdat that contained 3 rows the first 2 emails arrived at the desired inbox but the last one landed in a spam folder instead. One way to avoid this is to add delays between each email, or at least delays in between email to the same recipient, however this is not guaranteed to deal with all possible classifications as spam. The suggested revised code is specified below. The SLEEP(n,unit) function delays execution in a data step by n units. In the example below unit = 1 and this corresponds with seconds.

```
FILENAME mailbox EMAIL;
DATA _null_;
  SET email_list END=EOF;
  FILE mailbox;
  […]
  PUT "!em_send!";
  _DELAY=SLEEP(60,1);
  IF EOF THEN "!em_abort!";
RUN;
FILENAME mailbox clear;
```

Using EMAILSYS=MAPI significantly reduces the probability of the email to be classified as spam.

SUMMARY OF SAS METHOD

SAS provides a flexible tool for sending bulk emails through the use of directives, however as noted there may be unexpected problems with spam algorithms if an SMTP connection is used. When using the MAPI method it may be necessary to run Outlook first to make sure that all components are imported properly. This can be automated by SAS using call system or X command.

SENDING EMAILS USING VISUAL BASIC (VBA)

INTRODUCTION TO OBJECT ORIENTED PROGRAMMING (OOP)

Visual Basic is one of object-oriented languages. Performing a task in Visual Basic involves manipulating various types of objects, each of which may have several different properties and methods.

One of key factors in achieving efficiency in OOP is very good understanding of available objects and their characteristics. And in that respect it is not different from SAS 4GL where programmer also needs to get insight into the core DATA/PROC steps to become fluent.

There is one big difference between SAS 4GL and OOP though. In 4GL all DATA/PROC steps can be considered as standalone procedures, i.e. understanding one procedure doesn't require familiarization with other components. It is different in OOP. In OOP each object is instantiated based on the parent class. This means that a given object inherits all methods and properties of that class. At the same time, methods and properties may be instances of other classes and they may be nested further. OOP can be thereby considered as a system consisting of many levels of nested connections.

As OOP is a methodology used in many of modern programming languages (e.g. Python, C++, Java) and at the same time SAS platform allows for launching application written in other languages it seems very inevitable for a SAS programmer to start development of OOP skills.

INTRODUCTION OF OUTLOOK EMAIL OBJECT

As already mentioned OOP also requires a very good knowledge of existing classes so that the programmer doesn't need re-invent the wheel. The first step is to make sure that the Developer menu is available in Outlook 2010 (or later). To do this, go to File -> Options -> Customize Ribbon and check the "Developer" box in the right side window.

The following will assume a very basic knowledge of VBA and will start from defining object references to Outlook application and email as follows:

```
Dim OutApp As Object
Dim OutMail As Object
Set OutApp  = CreateObject("Outlook.Application")
Set OutMail = OutApp.CreateItem(0) [=OutApp.CreateItem(olMailItem)]
```

Provided that the code above is run in a built-in VBA editor for Outlook the user can add a watch for both objects to see their properties & methods (VBA editor -> Debug -> Add watch -> put name of object in this case). The screen similar to Display 1 should be seen:

Expression	Value	Type
OutApp	"Outlook"	Object/Application
Outmail	""	Object/MailItem
Actions		Actions/Actions
AlternateRecipientAllowed	True	Boolean
Application		Application/Application
Attachments		Attachments/Attachments
AutoForwarded	False	Boolean
AutoResolvedWinner	False	Boolean
BCC	""	String
BillingInformation	""	String
Body	"	String
BodyFormat	olFormatHTML	OlBodyFormat
Categories	""	String
CC	""	String

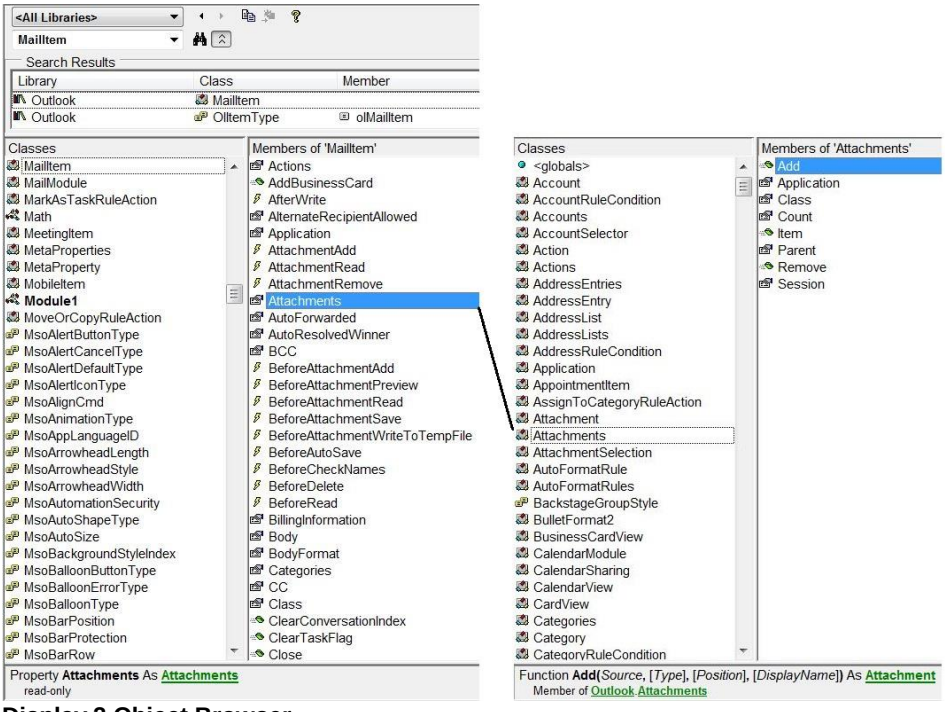
Display 1 VBA Watch – first glance at objects.

So what can be read from the watch window? For both objects it can be identified what type of object (class) it is – "Application" & "MailItem" respectively in the above presented example. As shown it is possible to review most of the properties/methods directly from the watch window. We can obtain the full picture by using the Object Browser.

OBJECT BROWSER

Object Browser can be accessed through: Developer tab -> Visual Basic icon -> View tab -> Object Browser. Display 2 illustrates functionality of the Object Browser. After we initiated email object ("OutMail") and found out that it is associated with "MailItem" class we are ready to explore the properties & methods of this specific class. One of the available properties is "Attachments". It is referenced as "Property Attachments (X) As Attachments (Y)". This means that it (i.e. "Attachments" property) is an instance of the class "Attachments (Y)". The diagram on the left side of Display 2 lists the available properties & methods for this class. For example, detailed syntax for the "Add" method is explained as: "Add (*Source*, [*3 optional parameters*])". Simple VBA code to add attachment to an email would therefore look as follows:

```
Dim email_attachment as String
email_attachment = "path\filename"
With OutMail
  .Attachments.Add (email_attachment)
End With
```



Display 2 Object Browser

VB SCRIPT VS. VISUAL BASIC FOR APPLICATIONS

Following the quick introduction to object oriented programming and Outlook objects we are ready for writing simple VB script which will send emails automatically based on email list.xlsx.
The application will be created as a standalone VB script (*.vbs file). It should be noted that there are some differences between Visual Basic for Applications (VBA) and VB script (VBScript) as VBScript is simply "lightweight" subset of Visual Basic. Examples:

- Definition of variables/objects: in VBA it is very common to use "Dim var1 as String" notation. It is not possible in VBScript which assumes all variables as Variant. In VBScript correct syntax is: "Dim var1"
- Definition of arrays: in VBA App it is quite common to write: "Dim array1(1 to 5, 1 to 10) as Integer – 5x10 array created. This will not work in VBS script. Instead we should use: "Dim array1(4, 9)" – here the index starts from 0, rather than 1 which SAS programmers may be used to.

DEFINITION OF KEY OBJECTS & VARIABLES

The following code:

- (1) creates instances of Outlook & Excel applications and file system objects;
- (2) produces variables pointing at paths to email list.xlsx, outlook.exe and a temporary location;
- (3) opens the Outlook application. The OutApp object will be created without unexpected disruptions if the Outlook application is opened correctly.

```
'***(1)***
Dim ExcApp, OutApp, Fso
Set ExcApp = CreateObject("Excel.Application")
Set Fso = CreateObject("Scripting.FileSystemObject")
'***(2)***
Dim inputfile
inputfile = "path\to\Email list.xlsx"
Dim outlookpath
outlookpath = "path\to\outlook.exe"
Dim desktoppath
desktoppath = "path\to\temp\location\"
'***(3)***
Set objShell = CreateObject("Shell.Application")
objShell.ShellExecute outlookpath
```

DELAYING EXECUTION OF VBA CODE

Outlook.exe will need several seconds to open properly so execution of further parts of the code needs to be delayed. The following code creates a temporary Windows .bat file calling the DOS timeout command. The file is subsequently executed using ShellExecute method and finally deleted.

```
'***(4)***
Set tempfile = Fso.CreateTextFile(desktoppath & "temp.bat", True)
tempfile.writeline "timeout 10"
tempfile.close
objShell.ShellExecute desktoppath & "temp.bat"
Fso.DeleteFile desktoppath & "temp.bat"
```

LOADING EMAIL LIST.XLSX INTO ARRAY

The following code opens "Email List.xlsx" (5), defines a 3x5 array called "all", loads the content of the input file into the array (6) and finally closes file and excel without saving changes (7)

```
'***(5)***
Dim ExcObj
Set ExcObj = ExcApp.Workbooks.Open(inputfile)
'***(6)***
Dim all (2, 4)
With ExcObj.Worksheets("Sheet1")
  Set Rng = .Range("A2:E4")
End With
Dim rowcnt
rowcnt = 0
For a = 1 To Rng.Rows.Count
  For b = 1 To Rng.Columns.Count
    all(rowcnt, b-1) = Rng.Cells(a, b)
  Next
  rowcnt = rowcnt + 1
Next
```

```
'***(7)***
ExcObj.Close False
ExcApp.Quit
```

LOOPING THROUGH ALL ROWS OF ARRAY

A For … Next loop is used to go through all elements (rows) of the array - for each iteration an email will be sent. The syntax is as follows:

```
For i = 0 To rowcnt - 1
  [code]
Next
```

The next sections will define the contents of "[code]".

ERROR HANDLING

Although execution of code was delayed to allow outlook.exe to fully run there still may be a risk of code failure due to incomplete launch of Outlook Application. A very common error is: "Run-Time error 429: ActiveX Components Can't Create Object". An error or exception handling rule should be implemented to deal with this. The following code uses an existing instance of Outlook if ready (through GetObject) (8) but if the compiler raises error 429 then a new instance of Outlook Application object is created (9). The error exception rule prevents the application from working unexpectedly.

```
On Error Resume Next
'***(8)***
Set OutApp = GetObject(,"Outlook.Application")
'***(9)***
If Err.Number = 429 Then
 Set OutApp = CreateObject("Outlook.application")
End If
On Error GoTo 0
```

SENDING EMAIL

First, an email object (10) needs to be created and variables defined for keeping information required to fill in all key fields of the email form (11). Step 11.1 requires additional details – a VBA split function dynamically creates an array of substrings from a given string expression using specified delimeter. For example, name = split ("Name Surname", " ") (0) produces the following array: {"Name", "Surname"} and assigns its first value to the variable "name". And finally, properties & methods of the email object are assigned values (12). If user is interested in sending emails in plain format step 12.1 can be modified as follows: .BodyFormat = 1 & .HTMLBody replaced with .Body.

```
'***(10)***
Dim OutMail
Set OutMail = OutApp.CreateItem(olMailItem)
'***(11)***
programmer = all(i, 0)
programmer_email = all(i, 1)
 '***(11.1)***
programmer_name = Split(programmer, " ")(0)
email_subject = all(i, 2)
email_customtext = all(i, 3)
email_attachment = all(i, 4)
If email_customtext <> "" Then
  HTMLBody = "Dear " & programmer_name & ",<BR><BR>" & email_customtext
Else:
  HTMLBody = "Dear " & programmer_name & ",<BR><BR>" & "Please review attachment."
End If
'***(12)***
With OutMail
  .To = programmer_email
  .Subject = email_subject
  '***(12.1)***
  .BodyFormat = 2
  .HTMLBody = HTMLBody
  If (Fso.FileExists(email_attachment)) Then
    .Attachments.Add (email_attachment)
  End If
  .Display
  .Send
End With
```

It is advised to comment out ".send" method (or put into IF … END IF block) until it is certain that the code works as expected.

SAS AND VBA CONNECTED TOGETHER

SAS code can use elements of VBA when using the DDE device type as the example below shows. The code opens an XLSM file containing a VBA macro of interest and executes specific VBA code ("Macro1" in example below is the name specified in the Sub statement).

```
options noxwait noxsync;
%let _file=path\to\xlsm\file\containing\VBA\macro\file.xlsm;
x "start excel /r &_file";
filename fileref1 DDE "Excel|system";
data _null_;
file fileref1;
put '[RUN("Macro1")]';
run;
filename fileref1 clear;
```

Alternatively we could use call system or X commands to run VBScript (.vbs macro) directly without needing to use obsolete DDE.

Are there any scenarios which would be ideal ground for such a hybrid approach? It is hard to find justification for mixed approach, unless you already have VBA programmed and you want to move some automation into SAS, e.g. to send emails after a SAS report has been created.

CONCLUSION

This paper has illustrated three methods for automated distribution of emails: SAS, VBA and a mixed approach. All methodologies fit their purpose and it is not legitimate to say that any of the methods is superior to others. A programmers' skill base will probably become the key selection factor, as well as software and tools already available, for example SAS and the Microsoft Office package.

REFERENCES

1. Erik W. Tilanus, "Using Mail Functionality in SAS®", SAS Global Forum 2013, Paper 023-2013
2. David H. Johnson, "E–mail from your SAS® Session: How and Why", SUGI 31, Paper 256-31
3. "Visual Basic: Objects and collections", http://www.hep.ucl.ac.uk/~za/current/vba/VBA_Objects.pdf

ACKNOWLEDGMENTS

Big thanks to William Greenway and Vishal Donda, who agreed to perform technical review of the paper, Shrishaila Patil for providing many helpful comments.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Piotr Karasiewicz
Quanticate International Ltd
Stefana Kazimierza Hankiewicza 2 street
Warsaw / 02-103
+48 507400909
piotr.karasiewicz@quanticate.com