

Paper 256-31

E-mail from your SAS® Session: How and Why

David H. Johnson, DKV-J Consultancies, U.K & Australia

ABSTRACT

While we sleep, or attend to other issues, SAS can run some routine processing for us. A SAS job can be instructed to send an e-mail to confirm its completion, to provide results, or to warn of particular issues. At some suitable time we can then read our e-mail for confirmation of the process, for information that is distilled from the process, or for data that requires verification. If our process handles 1,000 records daily, this e-mail process can help us to save time by drawing our attention to only the problems in that data.

This paper examines one such process as an example of "e-mailing exceptions". We look at a number of ways to send e-mail from the SAS process. The methods include a very simple approach, a generic macro approach, a batch e-mail process, and reporting exceptions from, of all things, our e-mail server.

For SAS® 8.2 and later, the techniques work on most computer systems, although we touch on the use of "sendmail" functionality on UNIX from SAS sessions that are running SAS® 6.12 or that have tighter security settings. We also touch on similar processes for mainframes that are running earlier SAS versions or that have security constraints. The product that we use is Base SAS®, and the code might be less familiar, but it is well within the reach of all levels of SAS users.

INTRODUCTION

This paper concentrates on the code that we write to send emails for a SAS® session. We will give the code context by dealing with three reasons for using email in our batch processes: "Quickly deliver results", "Manage exceptions", "Targeted reporting". With each reason, the code we use will become a little more complex and provide additional functionality. After we have looked at this code, we'll touch on some of the issues that can arise, including the functionality available on our computer systems.

QUICKLY DELIVER RESULTS

Our first example follows a request from our Sales department to report on the high value properties we have on our books. The data is held in the SASHELP.HOUSES table. (If you don't have this table in your SASHELP library, then either create the samples from your SAS/ASSIST menu or you can get the data step code from the SAS Support web site.)

```
Ods Pdf;  
  
Title "Properties for sale over $100,000.";  
Footnote "Run by &SysUserId using &SysVLong4 on &SysSCp &SysSCpl..";  
  
Proc Print Data = SASHELP.HOUSES NoObs;  
  Where Input( PRICE, Dollar12.) > 100000;  
Run;  
  
Ods Pdf Close;
```

This is a simple piece of code generating a PDF print of the houses for sale with values over \$100,000. Titling and footnoting give us a context for the report. Note that the record selection converts the house price from a character value to a numeric one so we can select by value.

At the end of the data step, we'll have a PDF file called "SASPRT.PDF" created with the results. (Note that while this code works in SAS9, Version 8.2 requires that a file name be assigned to the ODS Pdf destination.) Bring this window to the top and we should see a banner like that shown in figure 1 above the PDF.

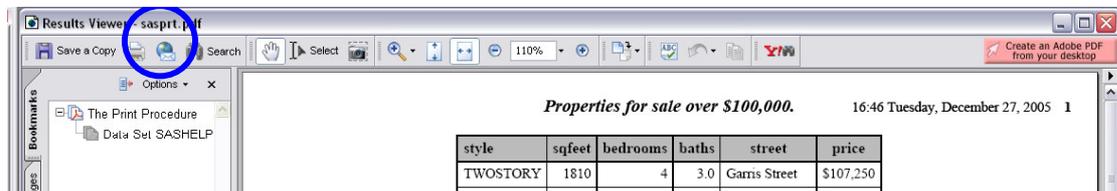


Figure 1

Note the symbol circled in blue. This allows us to email the file using the Adobe Acrobat Reader software. While useful, this is not the purpose of the paper.

Instead, select "File" from the SAS Window menu, and then "Send email..." which will bring up a dialogue like that shown in figure 2. We can fill in an addressee and a subject on the email.

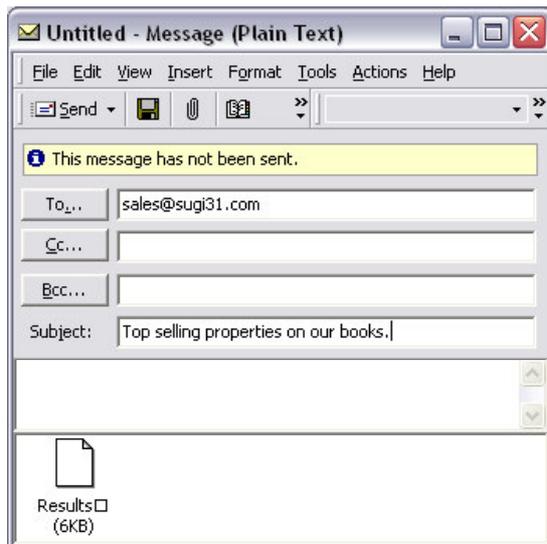


Figure 2

The file that is attached is a version of the results viewer and will not be directly viewable. That is because it is not SAS displaying the results to us, but SAS providing a window on an instance of the Adobe Acrobat reader. Acrobat masks the file name from SAS, so SAS calls the object "Results". When we receive the email, we could rename the attachment to "Results.Pdf" and it will then be viewable in Adobe Acrobat Reader.

Alternatively, we can attach the PDF file as a standard file attachment using the "paper clip" option available on Outlook (and similar functionality on other email clients.) That is most convenient if we want to also send the log from the job. By selecting the SAS log window first, the log window contents are attached automatically, and we can then select the PDF output file as a second attachment. The message is demonstrated in figure 3.

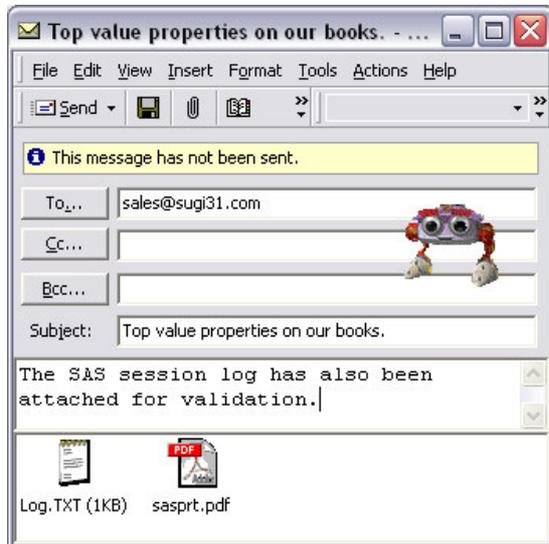


Figure 3

In figure 4, we can see the message in our Outbox folder.

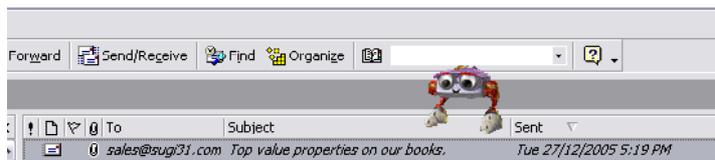


Figure 4

Later in this paper we'll look at using email clients other than Outlook, and change some SAS configuration settings. If the above has not worked for you, you may wish to skip ahead to "SAS messaging options" before trying other email examples.

MANAGE EXCEPTIONS

Sending email from selected procedures and data steps clearly has value if we want to share the results of our analyses and exploration of data. However, once we add email reporting to our batch processes, and share result sets from routine analyses, we start to see the power of email as a means to share intelligence on our business.

To send these results we need to find a programmatic way to send output from our jobs. The central idea of the email output process is that it is a special file to which we can direct results.

The file reference accepts additional parameters when it is used as an email destination, and we'll see some of these in action. To identify the file as an email destination, we use the supplemental parameter "email" in a manner similar to that shown here in a log excerpt.

```

220 FileName SENDMAIL Email "testme@sugi31.com";
221
222 Data _NULL_;
223   File SENDMAIL;
224   Put "Hello world.";
225 Run;

```

NOTE: The file SENDMAIL is:
E-Mail Access Device

```

Message sent
  To:          testme@sugi31.com
  Cc:
  Bcc:
  Subject:
  Attachments:
NOTE: 1 record was written to the file SENDMAIL.
      The minimum record length was 12.
      The maximum record length was 12.
NOTE: DATA statement used (Total process time):
      real time          0.90 seconds
      cpu time           0.04 seconds

```

The default parameter of the email file destination is the recipients' address. In the log excerpt above we can see the value is populated to this email parameter. To add other parameters we need only name them and include them on the filename statement or the file reference in the data sep. The following examples will demonstrate this syntax.

Let's demonstrate exception management by sending house details where the value is above certain parameters. We'll define the parameter for each property type in a format procedure step, like the following.

```

159 Proc Format;
160   Value $HighVal "RANCH"   = "85000"
161           "SPLIT"    = "90000"
162           "CONDO"    = "115000"
163           "TWOESTORY" = "100000";
NOTE: Format $HIGHVAL has been output.
164 Run;

NOTE: PROCEDURE FORMAT used (Total process time):
      real time          0.00 seconds

```

This will allow us to segment our business based on profitability, or identify listings that have a higher potential profitability and need special attention. An image showing the final message is shown in figure 5.

```

165
166
167 FileName SENDMAIL Email "vipteam@sugi31.com"
168           Cc = "salesdirector@sugi31.com"
169           Bcc = "audit@sugi31.com"
170           Subject = "High value properties for V.I.P. attention";
171
172 Data _NULL_;
173   File SENDMAIL;
174   Set SASHELP.HOUSES( Where = ( Input( Price, Dollar9.) >=
175                               Input( Put( STYLE, $HighVal.), 6.) ) );
176   Put "Place these properties on the high risk book, "
177       "and track them on the diary system.";
178   Put STYLE STREET PRICE;
179 Run;

```

```

NOTE: The file SENDMAIL is:
      E-Mail Access Device

```

```

Message sent
  To:          vipteam@sugi31.com
  Cc:          salesdirector@sugi31.com
  Bcc:         audit@sugi31.com
  Subject:     High value properties for V.I.P. attention
  Attachments:
NOTE: 12 records were written to the file SENDMAIL.

```

```

The minimum record length was 24.
The maximum record length was 81.
NOTE: There were 6 observations read from the data set SASHELP.HOUSES.
WHERE INPUT(Price, DOLLAR9.)>=INPUT(PUT(STYLE, $HIGHVAL6.), F6.);

```

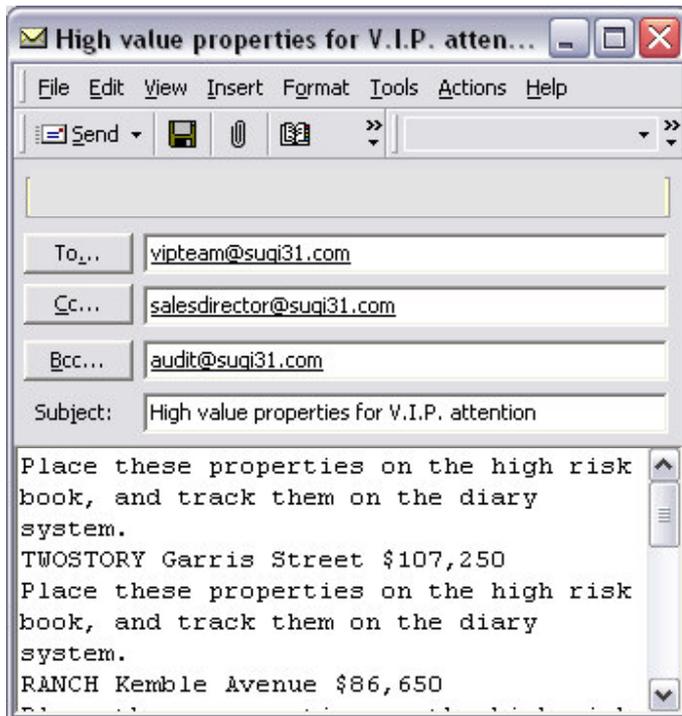


Figure 5

This message was expected to have a first line giving instructions for handling the properties, followed by a listing of all the properties. We did not get that result so in the following code we'll correct the output. For those with a lot of experience of output files, the mistake is obvious. For those less familiar with this form of output, can you see the change we made in the following email data step, and understand its implication?

As a first step, we'll create a PDF file to attach to our email. The PDF will be populated with a selection of the houses on our books. Note that we specified a physical destination for the ODS output, which is preferable for all batch code.

```

241 ODS PDF File = "c:\temp.pdf";
NOTE: Writing ODS PDF output to DISK destination "c:\temp.pdf", printer "PDF".
242
243 Data _NULL_;
244   File Print;
245   Set SASHELP.HOUSES( Where = ( Input( Price, Dollar9.) <
246                               Input( Put( STYLE, $HighVal.), 6.) ) );
247   Put STYLE /
. . .
253   _PAGE_;
254 Run;

```

NOTE: 45 lines were written to file PRINT.

NOTE: There were 9 observations read from the data set SASHELP.HOUSES.
WHERE INPUT(Price, DOLLAR9.)<INPUT(PUT(STYLE, \$HIGHVAL6.), F6.);

```

255
256 ODS Pdf Close;
NOTE: ODS PDF printed 9 pages to c:\temp.pdf.

```

Since we have directed our exceptional listings to the VIP team, in this report we will send the remainder to another part of the business. In this example we'll use the "file statement" options in the data step to manage our delivery, and we'll attach a set of pages that can be used by the team to manage individual properties. The final message is shown in figure 6. Part of the code is coloured blue and in bold face. We'll explore the reason for that after we have seen the result in the email.

```

257
258 FileName SENDMAIL Email "NUL";
259
260 Data _NULL_;
261   File SENDMAIL To = "salesteam@sugi31.com"
262             Cc = "dataquality@sugi31.com"
263             Bcc = "audit@sugi31.com"
264             Subject = "High value properties for V.I.P. attention"
265             Attach = "c:\temp.pdf";
266   Set SASHELP.HOUSES( Where = ( Input( Price, Dollar9.) <
267                               Input( Put( STYLE, $HighVal.), 6.) ) );
268   If _N_ = 1 Then Put "The following properties are reported in detail "
269                     "in the attached tracking file. Split thefile and distribute the "
270                     "pages to the agent handling each property.";
271   Put STYLE STREET PRICE;
272 Run;

```

NOTE: The file SENDMAIL is:
E-Mail Access Device

Message sent

```

To:          salesteam@sugi31.com
Cc:          dataquality@sugi31.com
Bcc:         audit@sugi31.com
Subject:     High value properties for V.I.P. attention
Attachments: c:\temp.pdf

```

NOTE: 10 records were written to the file SENDMAIL.
The minimum record length was 25.
The maximum record length was 155.

NOTE: There were 9 observations read from the data set SASHELP.HOUSES.
WHERE INPUT(Price, DOLLAR9.)<INPUT(PUT(STYLE, \$HIGHVAL6.), F6.);

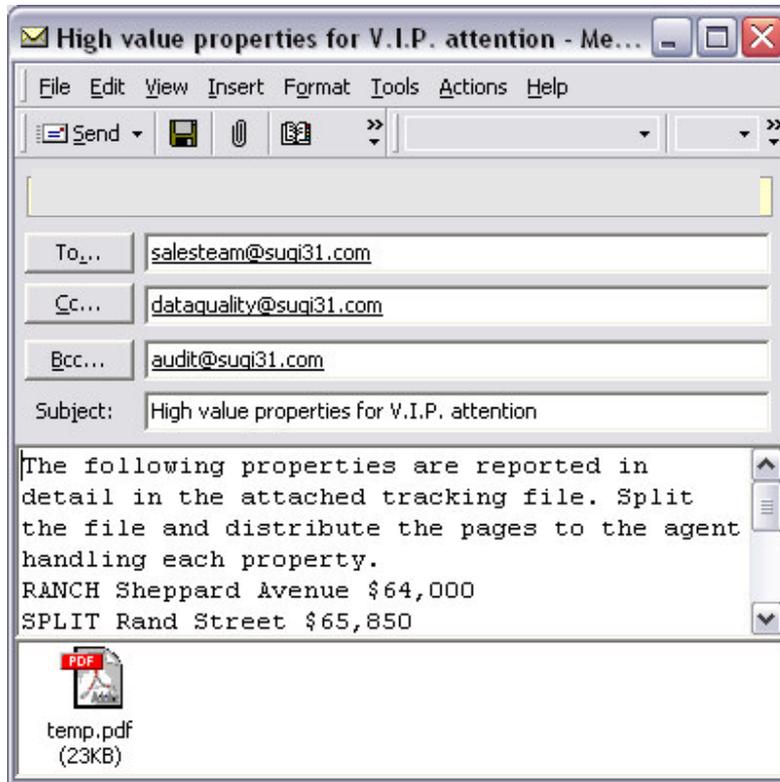


Figure 6

In figure 6 we can see the first instruction line is followed by the details of more than one property. The reason is that the statement to output the header line has been changed to only put the header line in the file on the first iteration of the data step. The statement is coloured blue in the log extract above, and the additional command in the statement is in a bold font.

The "NUL" parameter on the filename statement indicates that none of the email parameters are being set with the file reference, and that we will set them all in the data step. We may believe that if the data step does not iterate, then the email will not be sent. That is not the case, as the following example will demonstrate. The final email is shown in figure 7.

```

1   FileName SENDMAIL Email 'NUL';
2
3   Data _NULL_;
4   File SENDMAIL To = "noemail@sugi31.com"
5   Subject = "No data, so this email should not be sent";
6   Stop;
7   Run;
```

NOTE: The file SENDMAIL is:
E-Mail Access Device

Message sent
To: noemail@sugi31.com
Cc:
Bcc:
Subject: No data, so this email should not be sent
Attachments:

NOTE: 0 records were written to the file SENDMAIL.

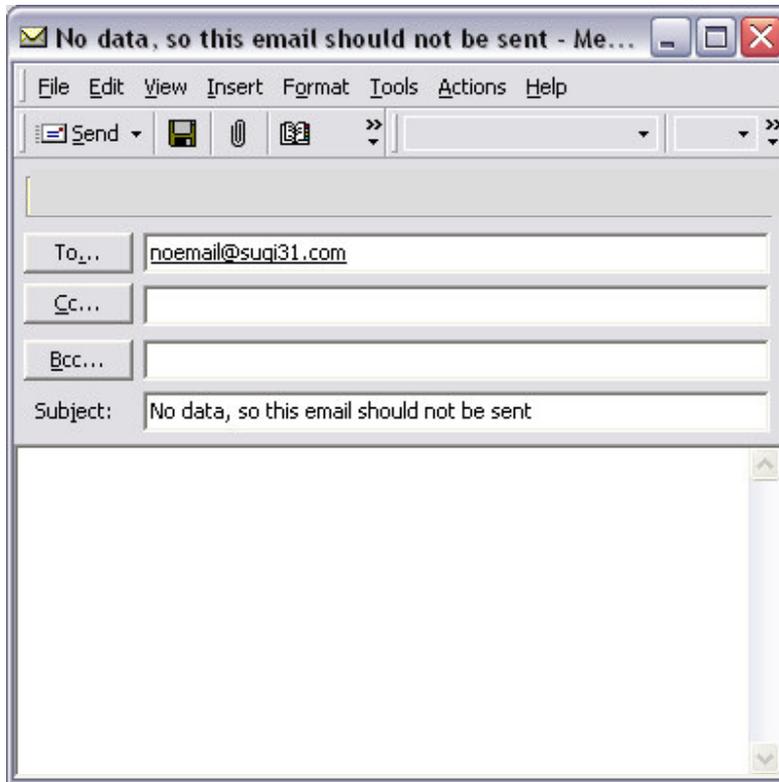


Figure 7

As the data step compiles, the file destination is opened, and then closed again on the completion of the data step. The opening and closing of the destination means an email will be created. To prevent an email from being sent in the absence of data, we will need to employ other techniques.

Consider the way the email parameters are set:

1. The parameters of the FileName command.
2. The parameters associated with the File statement in the data step. If defined, they will override the FileName command settings, which comprise a compiled global statement.
3. Any directive we can assign from data through iterating the data step will override the File statement, since the File statement is defined at compile time. We will look at data step directives in the next section.

If we can set the recipient address from data within iterations of the data step, this would allow us to set a default address we would use if there were no data to report. Where data is found, this would override these message parameters. In the absence of data, the data step directives would not be executed, and the FileName command and File statement assignments would take priority. That is why I prefer to set a default address on the FileName command, so that if there is no data, an email is still sent reporting missing data to the default address. We will explore this again in the next section.

TARGETTED REPORTING

The power of the SAS interface to email becomes most apparent when we use it to select exceptions and direct reports on these exceptions to particular parties. Rather than send copies of the same report to multiple parties, we can split reports, and send each section only to the party most interested in the report.

Let's demonstrate this by taking our SASHELP.HOUSES table, and send the records for each type of residence to specific email destinations. We have four property types, "Condo" which we'll report to the team looking after young first home-buyers, "Split level" which we'll report to our family team, "Two storey" properties which are handled by the "families with teens" team and "Ranch" which we'll send to the cowboy team. The simplest way to do this is to

associate each property type with an email address through the format procedure. Here is the code for those formats.

```
Proc Format;
  Value $StyleEml  "CONDO"      = "young@sugi31.com"
                  "RANCH"      = "cowboy@sugi31.com"
                  "SPLIT"      = "family@sugi31.com"
                  "TWOESTORY"  = "teens@sugi31.com";
Quit;
```

Now we want to process our data in house style order, so we can send one email to each team, with each email containing all the records for that team. In the following simple data step, we'll start by building the data set to write out the records. We'll write each group to the log so we can see how the messages will group.

```
Data _NULL_;
  Set HOUSES;
  By STYLE PRICE;
  If First.STYLE Then
    Put "Subject: Report on " STYLE " style properties" /
        "Send email to " STYLE $StyleEml.;
  Put "Address " STREET "Price " PRICE ", " BEDROOMS "bedrooms, "
      BATHS "bathrooms on " SQFEET "square feet.";
  If Last.STYLE Then
    Put "Send email now." //;
Run;
```

At the beginning of each group of house styles, we have identified the title or subject of the report, and the email address we want to use. Then we write the details of the properties. Finally, after the last record of the group we close and send the email. Here is a selection of the log that demonstrates the output.

```
Subject: Report on CONDO style properties
Send email to young@sugi31.com
Address Arcata Avenue Price $110,700 2 bedrooms, 2 bathrooms on 1860 square feet.
Address Jeans Avenue Price $127,150 4 bedrooms, 2.5 bathrooms on 2105 square feet.
. . .
Send email now.

. . .

Subject: Report on TWOSTORY style properties
Send email to teens@sugi31.com
. . .
Address Garris Street Price $107,250 4 bedrooms, 3 bathrooms on 1810 square feet.
Send email now.
```

The author makes it a practice to always test email data appearance by outputting the data to the log first. The format of the data lines can be checked, as well as the header and footer information put out with the emails and any groups.

CONVERT LOG TEXT TO EMAIL DIRECTIVES

Note the title line "Subject: Report..." that would be ideal for an email subject line. Note too the line that contains an email address. From our single table, we should be able to generate four emails programmatically. If we could associate the subject and email with a command that created the subject, and another command that assigned the email recipients address, then with a small change we can create our emails.

SAS provides that functionality through email directives that can be included in the output line. The directive for subject is "!em_subject!", and the one for recipient address is "!em_to!". The two lines of code that produced the subject and recipient lines in the above example can be replaced as follows.

```
/* Put "Subject: Report on " STYLE " style properties" /
```

```

        "Send email to " STYLE $StyleEml.; */
Put "!em_subject! Report on " STYLE " style properties" /
    "!em_to! " STYLE $StyleEml.;

```

Similarly, we may:

```

copy in another addressee in by using "!em_cc!",
modify our sender details by using "!em_from!", and
Attach a file by using "!em_attach!".

```

Note that there are some limitations on the last two commands we'll explore later.

As it stands, our code will build an email and send it at the end of the data step, just as we saw in the previous section. This is not quite what we want, so we need a command to send the email at the last record of each group. We want to replace the line where we output "Send email now". Here is how we will implement it in our code.

```

/* If Last.STYLE Then
    Put "Send email now." //; */
If Last.STYLE Then
    Put "!em_send!";

```

If we don't explicitly include the "!em_send!" directive, the email will be sent only once, instead of being sent at the end of each STYLE group. Before the end of this section, think about how that email might appear and be addressed and check on the answers that follow on page 13.

COMMAND DIRECTIVES

Since the completion of the data step produces an email, and our inclusion of an explicit "!em_send!" directive sends an email, we might expect that these data step changes will produce FIVE emails. The fifth is sent at the end of the data step, and is a duplicate of the fourth. To prevent this happening, we need to make two changes:

1. We need to know when the data step is processing the last record of the input table, so we use the "End =" option on the set statement.
2. Then we test for this option being true, and issue the "!em_abort!" directive to prevent the fifth email.

Here is how the code change and addition look.

```

Set HOUSES End = LAST;
. . .
If LAST Then
    Put "!em_abort!";

```

Since we are setting multiple email parameters for each message, we should be careful of missing or misdirected data. Essentially, we need a command that will initialise our email directives and prevent a setting from one email carrying to another. To release the directive values, the special directive "!em_newmsg!" is employed. Here is the way we would add that to the existing code.

```

/* If Last.STYLE Then
    Put "!em_send!"; */
If Last.STYLE Then
    Put "!em_send!" /
        "!em_newmsg!";

```

As you can see in the following log excerpt, the completion of each message send is followed by the re-initialisation of the email directives. So, we should expect that with these changes we would now be sending emails. If we run the code, we get lines similar to the following in the SAS log. What is wrong?

```

!em_subject! Report on TWOSTORY style properties
!em_to! teens@sugi31.com
Address Highland Road Price $102,950 4 bedrooms, 2.5 bathrooms on 1745 square feet.
Address Garris Street Price $107,250 4 bedrooms, 3 bathrooms on 1810 square feet.
Address Sanders Road Price $55,850 2 bedrooms, 1 bathrooms on 1040 square feet.
Address Fairbanks Circle Price $69,250 2 bedrooms, 1 bathrooms on 1240 square feet.
!em_send!
!em_newmsg!

```

```
!em_abort!
```

The problem is that while we can see that we have well formed directives, they are being sent to the SAS log and not to the email destination. The solution is simple. We need to add the file reference defining our output destination, and a "File" statement in the data step. The following changes demonstrate the code required.

```
FileName SENDMAIL Email "Nul";

Data _NULL_;
  File SENDMAIL;
```

Running the code again, we can expect to get messages like the following in our SAS log.

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

Message sent
  To:          young@sugi31.com
  Cc:
  Bcc:
  Subject:     Report on CONDO  style properties
  Attachments:
. . .
Message sent
  To:          teens@sugi31.com
  Cc:
  Bcc:
  Subject:     Report on TWOSTORY  style properties
  Attachments:
NOTE: 32 records were written to the file SENDMAIL.
      The minimum record length was 9.
      The maximum record length was 86.
NOTE: There were 15 observations read from the data set WORK.HOUSES.
NOTE: DATA statement used (Total process time):
      real time          1.87 seconds
      cpu time           0.24 seconds
```

As we explored in the last section, the "Nul" defined on the file reference identifies that a non-existent email address is being associated with it. When I test programs on my own domain, I prefer to use a real address that I won't create within my code. Then if I receive email to that address, I know that I have an issue with the definition of the email addresses in the data step. The following might be suitable.

```
/* FileName SENDMAIL Email "Nul"; */
FileName SENDMAIL Email "stepfailure@sugi31.com";
```

From this report, we can see that SAS is prepared to report a Blind Carbon Copy (BCC) address in use for our emails. However, we have no mention of such a directive above. We might assume the directive is "!em_bcc!", but a search of the SAS Support web site doesn't find this term. Let's try this directive, and in the process look at some of the errors we can see if we try to "hack" undocumented commands.

```
123      Put "!em_subject! Report on " STYLE " style properties" /
124          "!em_to! " STYLE $StyleEml.
125          "!em_to! salesdirector@sugi31.com";
. . .
137 Run;

NOTE: The file SENDMAIL is:
      E-Mail Access Device

ERROR: Invalid E-mail recipient.
FATAL: Unrecoverable I/O error detected in the execution of the data step program.
```

```

    Aborted during the EXECUTION phase.
Message sent
To:          young@sugi31.com !em_to! salesdirector@sugi31.com

```

In this example we copied the “!em_to!” directive and added the sales director so he could see all emails. It might appear that the issue here is that we misspelled the domain name “.som” on our email address. But this is misleading us since a closer examination of the log note shows us that the “!em_to!” appears within the recipient specification. If we correct this mistake, we get the following result.

```

171      Put "!em_subject! Report on " STYLE " style properties" /
172      "!em_to! " STYLE $StyleEml.
173      "!em_bcc! salesdirector@sugi31.com";
. . .
185 Run;

```

```

NOTE: The file SENDMAIL is:
      E-Mail Access Device

```

```

ERROR: Invalid E-mail recipient.
FATAL: Unrecoverable I/O error detected in the execution of the data step program.
       Aborted during the EXECUTION phase.

```

```

Message sent
To:          young@sugi31.com !em_bcc! salesdirector@sugi31.com

```

Now we have the corrected email address but we still have an email directive in the recipient line, highlighted with a bold blue font above. Part of the problem here is that SAS believes we have multiple recipients, but we have not specified them correctly. If an email address contains a space, it must be enclosed in double quotes. Where multiple addresses appear as recipients, the group must be enclosed within brackets. Since this will place spaces between the addresses, the correct way to specify the address is to use double quotes on each individual address, and then collect the group of addresses within brackets. Here is an example of that code.

```

Put '!em_to! ("email1@sugi31.com" "email2@sugi31.com")';

```

Note that while address specification with single quotes might work with the Outlook application as our email client, it does not work with all email interfaces. Similarly, while Outlook will allow addresses to be delimited with a comma or a semicolon, that isn't true of the SMTP destination. We'll look at alternative destinations, including SMTP in a section at the end of the paper.

We suspect SAS expected we specified multiple email addresses, but have not specified them in the syntactically correct way. Of course we didn't do that at all, we wanted to specify one address for each of two different directives. So our mistake is subtler than that.

Since SAS handles the directives as a “stream”, (that is it receives a command and then a parameter), it has no way of knowing where the end of the directive parameter lies unless we tell it. We do that by issuing a carriage return command, and SAS then knows the command string has ended. Here is the corrected code with the change highlighted in a bold-faced blue font, and the correct result.

```

196      Put "!em_subject! Report on " STYLE " style properties" /
197      "!em_to! " STYLE $StyleEml. /
198      "!em_bcc! salesdirector@sugi31.com";
. . .
210 Run;

```

```

NOTE: The file SENDMAIL is:
      E-Mail Access Device

```

```

Message sent
To:          young@sugi31.com
Cc:
Bcc:        salesdirector@sugi31.com

```

```
Subject:      Report on CONDO  style properties
```

The SAS documentation uses the syntax of putting each directive, and line of the message out to the file with a separate "Put" statement. We can do this a little more economically by using the Put directive "/" as you see above.

One last note on "!em_bcc!" however: it is not well documented because it is not available on all platforms. While it works on Windows, it does not work on Solaris with version 8.2, and probably on all Unices as well as MVS. The BCC comment is missing from the SAS log on Solaris, so this is a good indicator that the syntax is unavailable.

A similar issue exists with "!em_attach!" under 8.2 on Solaris where an error is issued in this form when you attempt to attach a file.

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

```
Message(s) received from the pipe command:
/path/path/SAS_8.2/utilities/bin/sasmailer does not support attachments. Mail not
sent.
```

```
ERROR: Email could not be sent
```

This particular error is issued because we have defined the "sasmailer" as our email engine through the "emailsys" system option. This option is explored in the last chapter. For now it is enough to recognise that on Solaris at version 8.2, the "sasmailer" script does not support either email attachments or aliases. (Our email directory may hold an entry for "Sales Director", which is translated to "salesdirector@sugi31.com" by our email client at the time of sending the message. The name is an alias for the email address, and requires access to the email address book, which is not supported by the "sasmailer" script.)

I should also offer another caution about trying to "hack" undocumented SAS commands. They can cause memory and Windows protection errors that will possibly affect your SAS session, and may even crash your Windows session. If you must experiment in this way, make sure you save your code before you submit it, and you should save and close all applications apart from SAS when you test.

HOW WOULD THAT EMAIL LOOK? ANSWER TO QUESTION ON PAGE 10.

An email directive supersedes an existing directive, it doesn't concatenate. So, we won't have an email with four "!em_to!" addresses, nor indeed will we have four subjects. As in normal data step processing, the value will equal the last assignment in the Data step execution. So the last recipient, who receives details on "TWOSTORY" properties, will be selected.

The most useful point I find about this is that while the directives for recipient, carbon-copy recipients, sender and subject are defined in the header of the email; they can be set at any time in the building of the data step. This can be helpful where we want to send seven records to the latest address, but the first four records may have been written to our original table with a now inactive address. Here is some example data.

	AGENT	STREET	COST
1	fred.jones@sugi31.com	Northumberland Ave	£50.00
2	fred.jones@sugi31.com	Old Kent Rd	£150.00
3	fred.jones@sugi31.com	Fleet Street	£200.00
4	fred.jones@sugi31.com	The Strand	£250.00
5	john.smith@sugi31.com	Maylebone station	£300.00
6	john.smith@sugi31.com	Trafalgar Square	£350.00
7	john.smith@sugi31.com	Mayfair	£400.00

Figure 8

In the following code, we write all seven records to the latest agent of record. Compare the data in figure 8 with the excerpt from the log below.

```

293   FileName SENDMAIL Email "nodata@sugi31.com";
294
295   Data _NULL_;
296     Set RECORDS;
297     File SENDMAIL;
298     Put "!em_to! " AGENT;
299     Put STREET COST;
300   Run;

```

NOTE: The file SENDMAIL is:
E-Mail Access Device

Message sent

To: john.smith@sugi31.com

. . .

NOTE: There were 7 observations read from the data set WORK.RECORDS.

It is also very helpful where our subject might contain an item count. Within an email client, we might want to open the email that has a subject line indicating 337 accounts are in error, before the one telling us of four errors. A subject line built on the last record will allow us to better target not only our reporting, but allow us to assist the reader with prioritising multiple reports. A very simple change to our data step can give us such a helpful subject on our email. The following log excerpt should demonstrate this concept.

```

. . .
363   If _N_ = 1 Then LOWCOST = COST;
364     SUMCOST ++ COST;
365     Put "!em_subject! " _N_ " addresses totalling " SUMCOST GBPound9.2
366         " ranging from " LOWCOST GBPound8.2 " to " COST GBPound8.2;
367     Put "!em_to! " AGENT;
368     Put STREET COST;
369   Run;

```

NOTE: The file SENDMAIL is:
E-Mail Access Device

Message sent

To: john.smith@sugi31.com

. . .

Subject: 7 addresses totalling £1,700.00 ranging from £50.00 to £400.00

So, the body of the email is built progressively in the data step. So the content of [the single email](#) in our previous question will be [all the data](#) we would have split out to each of our four addressees.

EMAILS ABOUT EMAILS

Since I run my own email servers, I have access to quite a lot of data about the incoming traffic. Some of the traffic is incomplete, that is, a remote email server will start to send me a message, but not complete the process. This may be because the remote server has been identified on a "black list" service. These services gather information from various sources and identify servers that have been compromised and are being used as open mail relays, or have been set up for the purpose of generating spam email.

When the remote server connects to my server, it sends its name and is identified with an ip address. My server then looks up the ip address in the remote black list service. With the increase in spam, and the increase in the number of email servers around the world, it may take an appreciable amount of time for the look-up to be completed, and for me to tell the remote mail server that I will receive the rest of the message.

SYSTEM MONITORING

In the meantime, the remote mail server may close the session and retry later on in the day. This is usually called "server time-out". What interests me however is whether I am getting all my email. So, from this process I would like certain information:

- How long is the black list look-up taking? And is this time affecting the number of incomplete messages?
- Who is trying to send me email, and are they being blacklisted?
- Are there times of day when I get a lot of incoming emails, and so have a lot of black list lookups, and as a consequence have more messages fail?
- Are attempts being made to use me as an open mail relay, and are these coming from particular areas or addresses?

As a system administrator, this information is vital to my monitoring the health and security of my systems. Some of the things I can do include:

- Changing or redefining my black list processes if they are becoming slow and are impeding email from people I know.
- Create emails to send to domain administrators if it seems their system has been hacked, and they are being used as a source of spam or other attack.
- Send email server users an email on a daily basis reporting the emails that have failed during the day so the user can identify message failures involving people we know.

PROCESS OVERVIEW

Some of these processes are reasonably complex, so I won't be putting up the code here. But here is a summary of the steps that can be taken. Based on the code we have explored above, the code to perform these functions is not hard to develop.

- Read the email logs, they are usually in W3C format, which is well documented in many places on the web.
- The email log comprises one or more data rows for each message id. Each data row reports a transaction between the sending and the receiving email server. Transactions include: identifying yourself, naming the email recipient, naming the sender, providing the subject, specifying the size of the email subject and the responses to each transaction from the other email server. For a valid transaction, the response is usually "OK" or a similar message.
- Sort the transaction data using the message id and date time.
- Transpose the data, so that a single data row holds all of the transactions for a single message.
- Count and drop all the complete messages.
- Divide the remainder into three groups:
 1. Messages blacklisted: the response from the black list service appears in the log
 2. Malformed messages: the receiving server will complain about the format of the message lines. Frequently such messages involve blocks of HTML designed to direct the recipient to a website where code is executed, a specific embedded image is extracted, or the address is then flagged as "active" and able to receive more spam.
 3. Messages (allegedly from our own domain to an external domain) that are testing for an open mail relay. A failed password, an invalid username and an IP address from outside our own network will usually identify these.
- For blacklist messages, we can summarise these by senders email address. If we have used an old email address for subscribing to information services on the web, or for posting to newsgroups, then we can exclude emails sent to these addresses. We can also exclude emails sent from domains where spam activity is high. The remainder then might include people we know, and we can now follow up their emails with a direct message.
- Malformed messages should probably be ignored completely. If the HTML was intended to identify "active" addresses, then any response from us will flag the address.

- As indicated earlier, open mail relays are often the result of a system administrator misconfiguring their email server, or of a domain being hacked. We might generate an automatic email to send the relay attempt back to the domain administrator so they can take action to close the relay. To do this we could perform a “lookup” on the IP address and find the administrator details. By using a tool called “whois”, which is well documented on the web, we can perform automated enquiries of the domain registries and retrieve contact details for an IP address. Then generating an email is a simple step. If we intend to generate and send the emails automatically, we can store details of the emails in a data set, and match to our table before sending. This would prevent our becoming a source of spam ourselves.

The following code will demonstrate some of the extra steps needed for any automated process. These code samples came from an application used to administer the volunteer registry for SUGI31.

The first sample demonstrates use of a “process tracking”, or audit address to receive emails from failed processes. If the recipient list is empty, the address and subject set on the file reference will be used for the email sent from the data step, and an email will identify the problem with the process.

```
FileName SENDMAIL Email "audit@sugi31.com"
  Subject = "Data integrity failure (empty to: list)"
  ReplyTo = 'volunteers@sugi31.com'
  From = 'volsystem@sugi31.com'
  Encoding = 'ASCII';
```

MESSAGE ENCODING

The file reference includes sender and reply addresses, which may be different. This encoding parameter overrides settings applied by Windows in its message defaults. The alternatives include “Mime/HTML” and “ISO-8859-1”.

The encoding option defines the way in which the message is formatted for sending. This is reported to the receiving message client in the header field “content-type”. The specification <Content-Type: text/plain; charset="iso-8859-1"> is the result of using a text or ASCII specification. This specification was very important where a hidden Windows setting was causing some of the author’s emails to be sent as “iso-8859-15” which was unreadable by some clients.

There is also a body of opinion that HTML should not be used for emails except where it is absolutely essential. Originally, the size of HTML messages, and its waste of bandwidth and dial-up time were the major reason for this opinion. Now, the use of HTML by spammers to verify email addresses and deliver malware to email recipients provides an even more compelling reason. Indeed, an exploit involving images in the WMF format (Windows MetaFile) was only found and patched in January of 2006. We must expect other vulnerabilities will be found in the future. Since ASCII encoded emails tend not to have these vulnerabilities, it is both conservative and cautious.

In the following code sample and email header, we see the addition of names (alias) to our email address. Observe the format, SAS recognises that a string delimited by “<” and “>” will be the email address, and will not object to the spaces embedded in the parameter of the directive. This syntax has been proven on clients other than Outlook, so it should be universal.

Note too the use of the “!em_from!” and the “!em_replyto!” directives to set these message parameters in the data step. These values have been taken directly from the header of a successfully sent email message.

```
Put '!em_from! SUGI Volunteer system <sugisystem@sugi31.com>' /
  '!em_sender! SUGI Volunteer CoOrdinator <sugivols@sugi31.com>' /
  '!em_replyto! SUGI Volunteer Registration <sugivolreg@sugi31.com>';
```

```
Date: 07 Nov 2005 01:08:32 +1100
Subject: GJ0004 interest from Vol Unteer
From: SUGI Volunteer system <sugivols@sugi31.com>
Sender: SUGI Volunteer CoOrdinator <sugivols@sugi31.com>
Reply-To: SUGI Volunteer Registration <sugivolreg@sugi31.com>
To: v.unteer@hisemail.com
MIME-Version: 1.0
Content-Type: text/plain;
  charset=us-ascii
```

CHANGING THE SENDER NAME

The “!em_from!” directive may not work as you expect. In this case, the SAS session logged directly onto an email server using the email name “sugivols@sugi31.com” and its password. If we have logged on to the email server with a different email user name, then the email will be sent with both email addresses in the header and the words “on behalf of”. It will look similar to the screen shot in figure 9. This behaviour closes a vulnerability in earlier SAS versions that would have allowed SAS to be used to send spam email with forged email headers. Note that this “on behalf of” behaviour has not been observed on Solaris with either version 8.2 or SAS9.



Figure 9

SENDING BULK EMAILS

If we use the MAPI interface to our email client (such as Outlook) to send emails, then sending large numbers of email from a data step does not seem to be a problem. However, some SMTP servers expect that a large number of emails sent in a short period of time are spam. Consequently, if we generate a number of messages too quickly we may find many were not received. There don't seem to be any systematic indications that emails were discarded, so the behaviour may go unnoticed, with our believing that the email recipients have just failed to answer our message. If we are using our email system to report hack attempts to system administrators, we probably expect already that the email may be ignored. That has been the authors experience with regard to all of the larger service providers. As a safety precaution, it is best the emails are sent at intervals when using the SMTP method. We might include the following in our data step.

```
If Not LAST Then Do;
  Put '!EM_SEND!' /
    '!EM_NEWMSG!';
  SLEEP = WakeUp( DateTime() + 30);
End;
```

Note again that since the file assignment sends an email on the last iteration of the data step, we don't need to explicitly send the message, and won't include a redundant delay on the last message. The nature of MAPI and SMTP clients is explored in more detail towards the end of this paper.

EMAIL BODY PARAGRAPHS

The structure of body text can be a little more difficult to format. Observe the results of the following data step. The first series of Put statements (coloured in blue) send 50 byte strings to the email destination, and SAS breaks the message line at the end of the 5th such string, or 250 bytes in length. The following statement (coloured in orange) builds a long string of 276 bytes, and attempts to send the whole string as one message line.

```
Data _NULL_;
  File SENDMAIL To = "longmessage@sugi31.com"
    Subject = "Testing paragraph wrapping in emails";
  Put " . _| . _1 . _| . _2 . _| . _3 . _| . _4 . _| . _5"
    " . _| . _6 . _| . _7 . _| . _8 . _| . _9 . _| . _0"
    " . _| . _1 . _| . _2 . _| . _3 . _| . _4 . _| . _5"
    " . _| . _6 . _| . _7 . _| . _8 . _| . _9 . _| . _0"
    " . _| . _1 . _| . _2 . _| . _3 . _| . _4 . _| . _5"
    " . _|* . _6 . _| . _7 . _| . _8 . _| . _9 . _| . _0";
  STRING = "The email client will split a line at a space " ||
```

```

"around 72 - 80 bytes. The Email file reference in " ||
"SAS forces a new line by byte 256, even though the " ||
"SAS V7 data set format allows column lengths up to " ||
"32k. This sentence will break at the * character " ||
"in this v*ery long paragraph.";
Put STRING;
Run;

```

As the screen shot in figure 10 shows, a break occurs at the 250th byte of the quoted string output, and the 256th byte of the literal "STRING". The two message break points are marked with blue ovals.

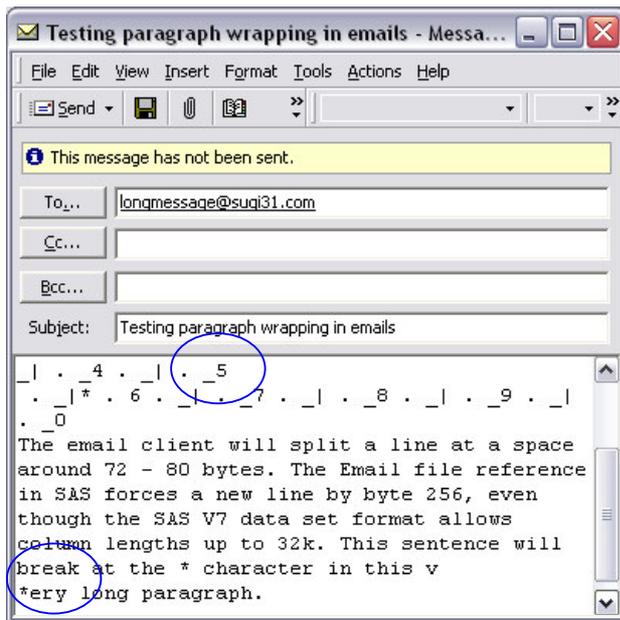


Figure 10

This difference means that we need to be careful of the way in which we construct paragraphs in the email data step. If our paragraphs are under 256 bytes then they will be output correctly. Otherwise, we will find sentences split as SAS structures the data to fit the destination.

If we were to send these text strings in separate emails, the message with the blue text would include the following log note.

The maximum record length was 250.

Similarly, a data step with the orange statement would include the following log line.

The maximum record length was 256.

So SAS advises us of the process it has executed, and it is up to us to be aware of the notes SAS includes in our logs.

SAS MESSAGING OPTIONS

The dialogue shown in our first examples is generated by my default Windows email client, which is Microsoft Outlook. The default client is defined in a registry setting in Windows. It can also be set in Internet Explorer through the menu item: Tools / Internet Options / Programs. By changing this value to "Outlook Express" my computer opens an Outlook Express message window. I also have "Pegasus Mail" installed as a messaging client, but when I select it I get the message window depicted in figure 11.



Figure 11

It is possible that if I had installed SAS on my machine after I had installed “Pegasus Mail” the “program hooks” to Pegasus would have been correct. It is more likely however that since Pegasus does not support a separate message window that it might still not support SAS messaging. To circumvent compatibility problems with an installed email client, or the absence of an email client on a system such as a server, there are SAS configuration options we can use. Most of them are “system invocation” options, which means we can read the value at any time, but cannot set them except at the start of our session through the SAS CFG file, or on the command line that starts our SAS session.

When I am using the external email client, this indicates that the following settings exist on the SAS session.

```
1   %Put %SysFunc( GetOption( EMAILSYS ) );
MAPI
2   %Put %SysFunc( GetOption( EMAILDLG ) );
Native
```

With the email system set to “MAPI”, SAS will use the available “Messaging Application Program Interface”, which is the Windows link to messaging. With the dialogue (EMAILDLG) set to “Native”, the SAS session will attempt to use the default application, which has been Outlook. If we change this to “SAS” in our configuration file, SAS will generate its own messaging window, which appears in figure 12. Note that “EMAILSYS” is a dynamic option, and “EMAILDLG” is an invocation option.

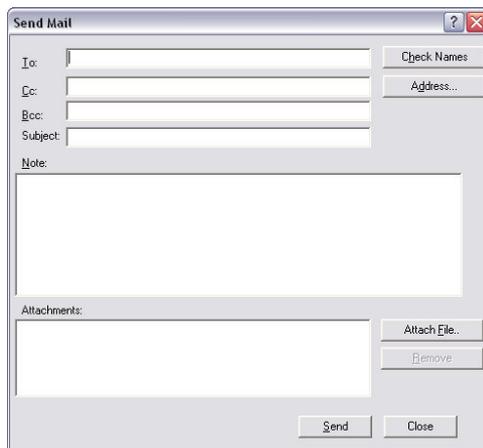


Figure 12

To make the change, look for the following option in the SAS configuration file. The file is usually called SASV8.cfg for version 8.2 and SASV9.cfg for the SAS9 versions. You may not find the option. The default setting is “Native” which means the entry is not required in the configuration file for the default behaviour. In that case add the line to the file and restart the SAS session.

```
-EMAILDLG SAS
```

Note that this change only affects the dialogue you use to create and send email. There is still a requirement for a suitable default messaging application to be available to the operating system. When we use the “Pegasus Mail” client as our default Windows email client, SAS still has no “program hook” available to check addresses or post its email. However, the SAS dialogue removes the default “active window attachment” behaviour that caused some

confusion when we reviewing the PDF results window. This comes at the cost of some of the default Outlook functionality.

ALTERNATIVES TO WINDOWS MESSAGING

An alternative to MAPI is SMTP, or Simple Mail Transport Protocol. This is the method used to send emails between Post Offices and allows us to bypass the restrictions of MAPI. However, this is only suitable if an SMTP service is running on your machine, or you can directly access your Post Office through SMTP. This functionality is usually blocked by most companies running email servers because of the risk of "spam" email being generated within the company.

To use the SMTP server, your server must either be running as an open mail relay, or you need to provide your username and password to the SAS session in the configuration file. (Actually, there are some operating system configurations that allow you to use your system authentication to access an email client, or email server. However, these require substantial systems authentication knowledge and are well beyond the scope of this paper, and probably not available to most users.)

"Open Mail Relays" are highly undesirable since they forward mail on behalf of anyone who sends them a message. Since any and all messages are forwarded, emailers of spam are exploiting "Open Mail Relays" around the world. If you run an "Open Mail Relay", this will quickly be identified, and your server will be added to a black list. Once you are on the black list, many legitimate email servers will then block all email from your email server.

The alternative, adding your username and password to your configuration file should only be done if you can conceal the configuration file in a place where any other user of the machine cannot find it. Here is the full set of options available for the SAS "configuration" file.

```
-EMAILSYS SMTP
-EMAILDLG SAS

-EMAILAUTHPROTOCOL LOGIN
-EMAILHOST mailgateway
-EMAILPORT 25

-EMAILID sugiemail@sugi31.com
-EMAILPW topsecret
```

- EMAILAUTHPROTOCOL specifies the type of authentication to be used for the connection to the server, and is apparently only available from SAS9 onwards.
- EMAILHOST is the name of the email server. This might be a fully specified domain name if you are accessing a service over the Internet. This name should resolve to an IP address. If you cannot "ping" this name, then you are unlikely to be able to send email through it. This is a system invocation option.
- EMAILPORT is the port that the server uses to receive email. The usual port is 25, but some services can set other ports such as 49 or 2525. Googles' "Gmail" service sets an obscure port for its SMTP access in an attempt to reduce its vulnerability to hackers. This is a system invocation option.
- The EMAILID is your username, and your password is saved in EMAILPW. It's worth remembering that anyone who can read these values from the configuration file can then send email as you through your email server. So be careful with the file. On a personal computer the risk is reduced if you place the configuration file in your "documents and settings" folder. These are system invocation options.

While some of the email options appear to allow us to produce emails with forged address details on certain systems, it is worth noting that an examination of the email headers reveals the genuine source of the emails. The SAS system is not a tool for sending Spam email. Indeed, one client has configured their Solaris server to apparently function as an open mail relay, making it possible to provide differing sender details for batch reporting. However, the system authentication bars the casual user, and the email server will only relay mail sent through the localhost.

PRIOR TO VERSION 8

Finally, the availability of the FileName engine for Solaris and MVS was a development for Version 8. It also depends on certain other system configurations that may not be available. So we'll close this by noting the methods employed by the author in Version 6 to send email from these platforms.

In an earlier section we touched on the process by which two email servers communicate. A series of messages are exchanged that permit a message to be passed. Prior to the filename engine being available, we could build a file containing our half of this series of exchanges, and then pass it to a system process that would open a communications socket with a remote server and pass each of our message parameters. It didn't have to interpret our data, and would only respond send each of our message lines if a positive response were received from the other end. If a negative response were received, it would close the communications channel and return a system fail code to us. As such, quite a simple system object would be employed, but we needed to produce well-formed commands if the message was to be exchanged.

The following lines of code would build a message file for that process.

```
PUT "HELO MYMVS";
PUT "MAIL FROM:<sasbatch@{MYMVS}>";
PUT "RCPT TO:<reportuser@sugi31.com>";
PUT "DATA";
PUT "FROM:sasbatch@{MYMVS}";
PUT "TO:< reportuser@sugi31.com >";
PUT "SUBJECT: Job 307: Report on key data.";
PUT "POSTED: 20060126 10:00";
. . . message body . . .
PUT // "NOTE: Do not use 'REPLY TO SENDER' option, it will not deliver correctly.";
PUT ".";
PUT "QUIT";
```

The "HELO" command in the first line, opens the communication line with the remote host, and includes the name of our own server. The following two lines identify the sender and recipient, and allow the remote server to close the channel quickly if the sender is blocked, or the recipient does not exist.

The lines between "DATA" and "POSTED" provide the header of the message, and the line that contains "." On its own closes the data part of the email. The "QUIT" command closes the proc (application) on MVS and may not be required on other systems.

To send this file, we need to pass it as a parameter to an application that will create the communication socket and pass the file. On Unix we might use "sendmail", which is well documented on many web resources. On MVS we will need a "step" in our job that calls an appropriate "Proc" which will route each of our data lines to the remote server. Lines like the following might be used.

```
//USERIDCC JOB ,',MSGCLASS=X,CLASS=C,REGION=15M,NOTIFY=USERID
//MYSASJOB EXEC SAS
. . . JCL lines for our SAS job . . .
//MAILOUT EXEC PCMAIL,COND=(0,NE,MYSASJOB),
// FILE='USERID.PDS.EMAILDATA(MYSASJOB)'
```

The "MAILOUT" line calls a registered "Proc" on our mainframe, and passes the input file we created from our SAS step. Note the "COND" parameter: it specifies that if the step called "MYSASJOB" finishes with a completion code "NE" to "0", then the step is not to be executed. This ensures emails are not sent unless the SAS session finishes cleanly.

While this process may seem to be rather tedious, and it relies on the development and maintenance of appropriate settings on the mainframe, it has one substantial advantage. Since a system "Proc" must be called, our Access Control Facility (RACF) can provide strict limitations on the users who can call it. This has the benefit of limiting the amount of email generated, and severely restricts the likelihood that our email system can be hijacked or abused.

CONCLUSION

In the section "Quickly deliver results", we looked at interactively using the email dialogue to send an email from the SAS session, and to attach a file or session results to the email.

"Manage exceptions" introduced the file destination to send email from a body of code. We saw it was possible to select data and write it to the body of the email within the data step.

"Targeted reporting" introduced the email directives that allow us to specify our recipients, subject file or attachment from the data we process in the data step. We also looked at the generation of multiple emails from a single data set and data step, deriving the address and subject details from the input data.

"Emails about emails" looked at some of the code elements we needed to pull together if we were generating email from our email server logs. In particular we explored some of the business imperatives that would drive us to write emails about emails. Our sample code included code that managed one of the many support systems underlying SUGI, and some techniques for managing large volumes of email, both by spreading it out, and by adding a better name to our email headers. We also touched on some of the problems of formatting the body of an email.

Our last section, **"SAS Messaging Options"**, looked at some of the SAS session options to change the integration of SAS with an email server, and an email client.

REFERENCES

Changes and Enhancements in Release 6.10 (SAS Institute, Cary), 55107, experimental, pp30 - 37 "electronic mail"
Changes and Enhancements for SAS under Windows, Release 6.11 (SAS Institute, Cary), 55276, pp 15, 66-72
"electronic mail"
Changes and Enhancements to Software, Release 6.11 (SAS Institute, Cary), 55300, pp 270-275 "e-mail"
Companion to the Microsoft Windows environment, Version 6 2nd edition, 55268 (SAS Institute, Cary), pp 79-83
"electronic mail"
SAS System Online documentation, V8 and later (SAS Institute Cary)
"Electronic mail - delivery solutions for information sharing", SUGA 1997 (Johnson)
"Just the facts ma'am' meets 'is it safe?'" SUGI 2001 (Johnson)

ACKNOWLEDGMENTS

The documentation available from SAS Publications, and on line in the SAS System is an invaluable reference. The people I worked with who developed and maintained the systems infrastructure, and provided encouragement. David Langton for his patience and time spent reviewing the paper and discussing concepts. As ever; my family for patience and support.

CONTACT INFORMATION

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

David Johnson
DKV-J Consultancies
Moorabbin Vic 3189 Australia
Phone: +44 7005 98 0828
+61 3 9553 0301
Fax: +44 7005 98 0829
Email: sugi31@dkvj.com
Web: <http://www.dkvj.com>

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.