Paper 3233-2019

Building a Custom Email Method: A Case Study

David L. Ward, Health Data & Management Solutions, Inc.

ABSTRACT

The email access method has long been a powerful tool for SAS® programmers, providing them with the ability to easily generate notifications or deliver content as part of their data processing tasks. Although the method does provide a bevy of options, it has a few limitations such as the inability to specify multiple senders, custom headers, or custom formatting of recipient or sender names. This paper guides the user through the process of bypassing the built-in email access method by writing a custom email method that writes SMTP statements directly. Not only do readers gain the ability to work around the stated limitations, but they also gain a deeper understanding of advanced programming techniques available to SAS such as the SMTP protocol, sockets, encoding, and encryption.

INTRODUCTION

The purpose of this paper is to present an alternative method of sending email using SAS which satisfies several requirements that are not possible using the built-in email access method. Along the way, the reader will gain a more in-depth understanding of how email is generated as well as learn some advanced file and communication techniques including sockets and file encoding. While SAS programmers of any skill level can benefit from this paper, those with at least a cursory experience with the email access method will be better prepared to fully grasp the concepts and put them to use.

BACKGROUND

Access Methods

An access method is a way of instructing SAS how to read and write data to some device or network address as if it were a regular file. The most basic access method with which all programmers will be acquainted – at least implicitly – is the *file* access method. This is the default access method which is specified when no access method qualifier is supplied in the filename statement. Access methods are available to allow reading and writing from SAS catalog entries, the system clipboard, FTP/SFTP, network sockets, and websites. In the following example, the website www.sas.com can be read just like any other local file on disk by using the URL access method:

```
filename website url 'http://www.sas.com';
data _null_;
  infile website;
  input; put _infile_;
stop; run;
NOTE: The infile WEBSITE is:
      Filename=http://www.sas.com,
      ...
      Service IP addr=23.67.205.217,Service Name=N/A,
      Service Portno=443,Lrecl=32767,Recfm=Variable
<!DOCTYPE HTML>
```

In this example you can see that what looks like a very straightforward data step with an infile statement actually reads the HTML source code typically streamed to a browser. In like manner, the email access method can be used to set up a filename to which an email message can be *written*:

```
filename msg email 'recipient@localhost' from='sender@localhost'
sender='sender@localhost' subject='Msg Subject';
data _null_;
  file msg;
  put 'Hello, this is a test message';
run;
NOTE: The file MSG is:
      E-Mail Access Device
Message sent
      To:          "recipient@localhost"
      Subject:     Msg Subject
NOTE: 1 record was written to the file MSG.
```

Notice the NOTES printed to the SAS log which indicate that the email access method was used, and that the message was sent successfully. Consult the SAS Language Reference Dictionary for a full list of various options which can be used with this access method.

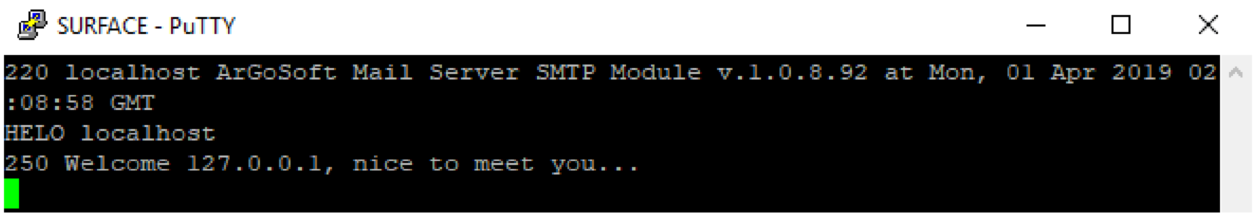
Email Access Method Interface Methods

The email access method supports three different interface methods. The interface method and method options are set using system options. The Windows default setting is MAPI or Messaging Application Program Interface which will attempt to use the system default application for sending email like Outlook, much like a browser may try to do when you click on a mailto: link. A second and more esoteric option is VIM or Vendor Independent Mail which is used for Lotus Notes or cc:Mail. This paper will make use of the SMTP or Simple Mail Transfer Protocol method. This is by far the industry standard method for sending and receiving email which has been around since 1982. SMTP is built into Unix and Linux distributions, and is an available installation option in Windows. It is a straightforward text-based network protocol transmitted over sockets, typically at port 25. This makes it relatively easy to understand and to create our own SMTP messages.

WHAT IS SMTP AND WHAT ARE SOCKETS?

SMTP – Simple Mail Transfer Protocol.

Email as we know it began in the 1960s. Various standards guided the communication etiquette – or protocol – for how client and servers were expected to "talk" to one another, and in 1982 the official SMTP RFC (Request for Comments) was written and widely adopted. For the interested user, you can read the document here: https://tools.ietf.org/html/rfc821 You can even connect to an SMTP server using TELNET in order to see the server in action. In this example, the Windows telnet program PuTTY is used to connect to port 25. The opening command HELO is sent and the server responds with a 250 message (OK).



```
220 localhost ArGoSoft Mail Server SMTP Module v.1.0.8.92 at Mon, 01 Apr 2019 02
:08:58 GMT
HELO localhost
250 Welcome 127.0.0.1, nice to meet you...
```

SMTP is a text-based protocol limited to lines of length 80 (including the CRLF termination sequence for a line). This means that it is not suitable for transferring binary data without encoding. This paper will explain encoding later on when demonstrating how to send attachments.

Network Sockets and the SOCKET Access Method

A network socket is a virtual representation of what used to be a *physical* port or socket. Much like a receptacle for power or a networking cable, a socket is a particular location where data is to be sent and received, available at a particular "port" number. In order to reach a particular computer on a network, an IP (Internet Protocol) address is typically used, thus a full network socket consists of a computer's IP address and a particular port number in the form <ip-addr>:<port-number>. If an IP address is like a street full of homes, the port number is the particular mailbox you are trying to locate. IP addresses are linked to domain names via something called DNS. In this paper, we will consider a network address to be either a name or domain name or an IP address. Over time, standard Internet Protocols became associated with standard port numbers, and the port number for SMTP be25. There are additional standard ports for SMTP which are used for things like encryption, or when port 25 is blocked by an ISP (Internet Service Provider) but for now simply consider SMTP to available on port 25.

One of the access methods mentioned earlier is the SOCKET access method. This method allows SAS code to read and write to a network socket as if it were a local file, simplifying syntax. The SMTP access method is actually similar to the SOCKET access method; it connects to an SMTP server using network sockets and formats an SMTP message for you under the covers. Later in this paper we will examine the SMTP that SAS builds using the DEBUG option. To see the socket access method in action, consider the following code:

```
filename mysocket socket '127.0.0.1:25' termstr=crlf;
data _null_;
  infile mysocket; file mysocket;
  input; put 'HELO localhost';
  input; putlog _infile_;
stop; run;
```

This code connects to the SMTP server at the IP address 127.0.0.1 (a special address representing the local computer, or the domain name localhost) using the termination string CRLF (carriage return, line feed). The data step connects at the input statement, then writes the HELO message just like our example above using PuTTY, and shows is the line that the server returns:

```
NOTE: The file/infile MYSOCKET is:
      ...
      Peer Hostname Name=localhost,
      Peer IP addr=127.0.0.1,Peer Name=N/A,
      Peer Portno=25,Lrecl=32767,Recfm=Variable

250 Welcome 127.0.0.1, nice to meet you...
```

CASE: LIMITATIONS OF THE SAS SMTP EMAIL ACCESS METHOD

Now we come to the original problem that necessitated writing a custom email method to begin with. While the SAS email access method has many convenient and powerful options, there are at least two which were not supported and were required for a particular application:

1. Supplying custom SMTP headers
2. Including a comma in sender or recipient address display names

SUPPLYING CUSTOM SMTP HEADERS

The message transmitted over SMTP consists of two parts: a header and a body. The headers are subject to grow as the message is passed from server to server. Most email clients will let you see the "source" or "original" message. Examine any of your email messages and see if you can look at the full underlying message data. You will see many interesting SMTP headers which show how the message traveled through the Internet, which kinds of security scans were performed, anti-spam protection, etc. We were required to add a custom header in order to instruct an upstream SMTP server that a message was automatically generated from an application so that it would not be subject to certain rules, like added sender or footer information.

Here is an example of the SMTP headers generated from the first test message we sent on page 2: *(headers ADDED by the SMTP program and not SAS are in cyan)*

```
Received: from [127.0.0.1] by localhost (ArGoSoft Mail Server .NET
v.1.0.8.92) with SMTP (HELO localhost)
    for <recipient@localhost>; Sun, 31 Mar 2019 21:49:20 -0500
Date: 31 Mar 2019 21:49:20 -0500
Subject: Msg Subject
From: sender@localhost
Sender: sender@localhost
To: recipient@localhost
X-Mailer: 9.04.01M4P110916
MIME-Version: 1.0
Content-Type: text/plain;
        charset=iso-8859-1
Content-Transfer-Encoding: 8bit
Message-ID: <4esfvyh01nfen0bw31032019094920@LOCALHOST>
X-FromIP: 127.0.0.1

Hello, this is a test message
```

As you can see from this example, our SMTP server is not adding very much because it is a simple shareware application (there will be more details and screenshots on this later). SAS generated all of the headers you see above. There is a convention of using "X-" as a prefix for custom SMTP and HTTP headers. Let's say we need to add a customer header of X-MyApplication: True. The headers would look like this:

```
Content-Transfer-Encoding: 8bit
X-MyApplication: True
Message-ID: <4esfvyh01nfen0bw31032019094920@LOCALHOST>
```

SAS does not provide a way to alter the headers in this way using the email access method.

INCLUDING A COMMA IN SENDER OR RECIPIENT ADDRESS DISPLAY NAMES

The built-in email access method cannot handle quoted display names in email addresses properly. The documentation indicates that addresses should be supplied in the format user@host or First Last <user@host>. Since display names can contain apostrophes or commas, double quotes are an accepted standard which most email clients can interpret properly. Thus "Last, First" <user@host> is an acceptable address, allowing a comma to be used in the display name and preventing the client from interpreting that comma as separating multiple addresses. When we attempt to use this as a recipient in SAS, it sends the address along without the double quotes, and since a comma is used to separate email addresses in lists, it is interpreted as two addresses. SAS will allow us to use this format in the SENDER address, but using it in the FROM address (which email clients use to display where the email originated) produces a hard error from SAS itself:

```
filename msg email ... from='"Last, First" <sender@localhost>' ...
S: MAIL FROM:<Last, First>
R: 501 Syntax Error: Unbalanced angle brackets
ERROR: Email: 501 Syntax Error: Unbalanced angle brackets
```

This log shows both the SMTP server response (501) and how SAS mistakenly re-formatted the address to <Last, First> and lost the sender@localhost portion. Without resorting to a custom email method as outlined in this paper, no suitable method for including a comma (or other potential custom formatting) in the FROM address was identified.

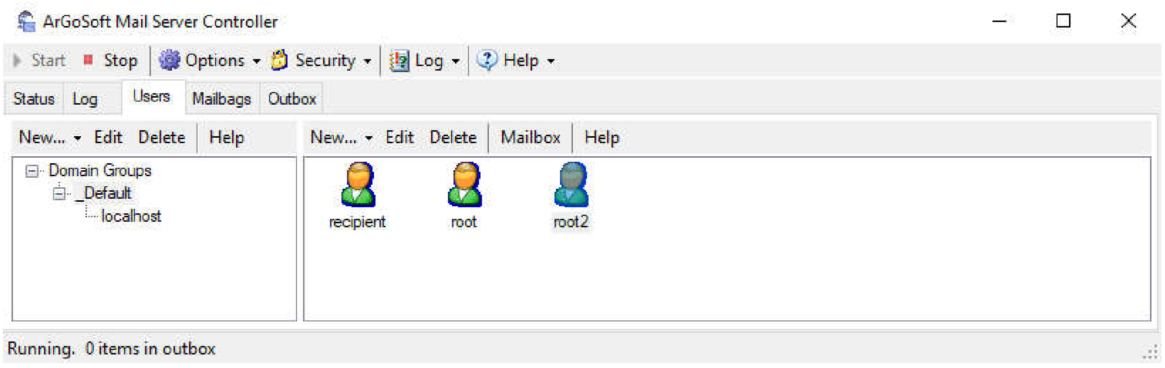
STUDY: TOOLS FOR EXPLORATION

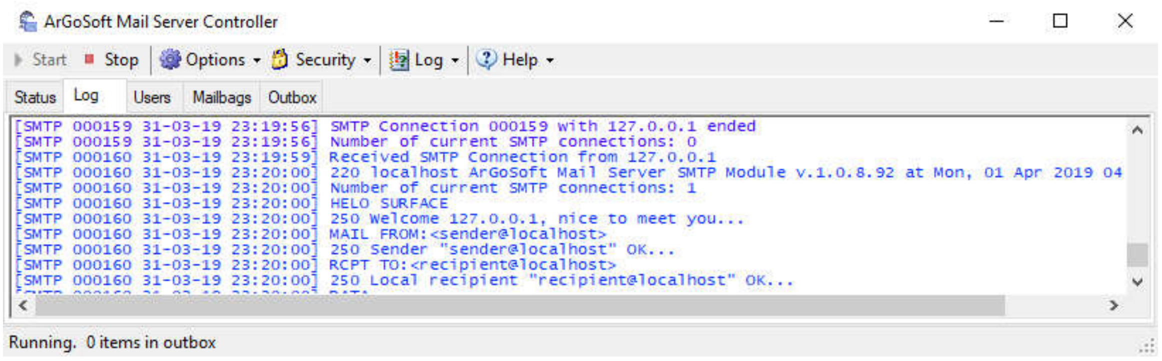
In order to properly write and test a custom email method, you will need the following applications in addition to the SAS System:

- An SMTP server
- An email client which can parse local raw email files
- An understanding of the DEBUG option in the SAS email access method

SMTP SERVER

We have chosen to use ArgoSoft Mail Server, a low-cost server for Windows available with a 30-day trial. This application is simple, intuitive, and provides just what we need – SMTP logging, mailbox delivery, and access to raw email messages once they have been accepted for delivery. In the following examples, a host of "localhost" has been set up with three valid users. The log tab shows the server accepting messages and interfacing with a client (in this case our sample SAS code):





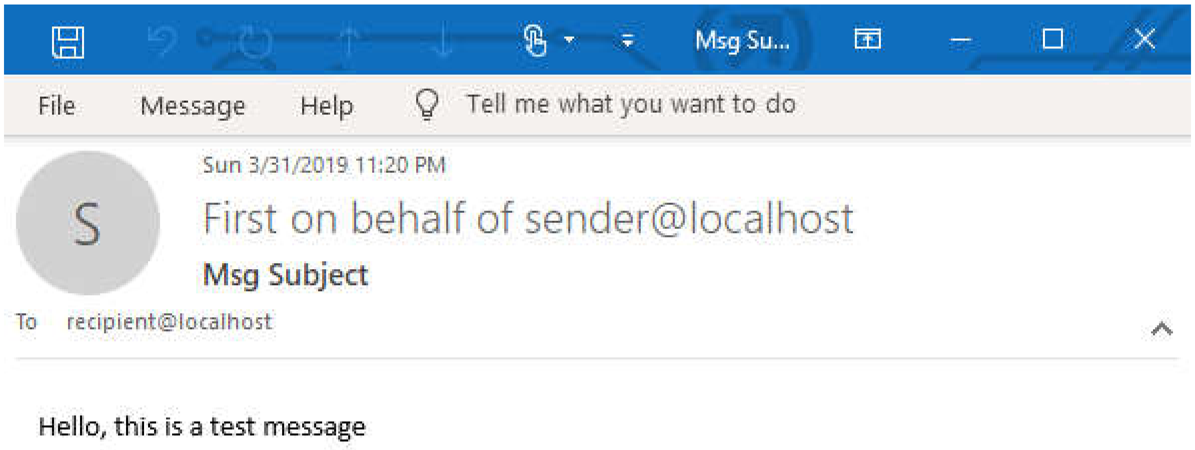
```
[SMTP 000159 31-03-19 23:19:56] SMTP Connection 000159 with 127.0.0.1 ended
[SMTP 000159 31-03-19 23:19:56] Number of current SMTP connections: 0
[SMTP 000160 31-03-19 23:19:59] Received SMTP Connection from 127.0.0.1
[SMTP 000160 31-03-19 23:20:00] 220 localhost ArGoSoft Mail Server SMTP Module v.1.0.8.92 at Mon, 01 Apr 2019 04
[SMTP 000160 31-03-19 23:20:00] Number of current SMTP connections: 1
[SMTP 000160 31-03-19 23:20:00] HELO SURFACE
[SMTP 000160 31-03-19 23:20:00] 250 Welcome 127.0.0.1, nice to meet you...
[SMTP 000160 31-03-19 23:20:00] MAIL FROM:<sender@localhost>
[SMTP 000160 31-03-19 23:20:00] 250 Sender "sender@localhost" OK...
[SMTP 000160 31-03-19 23:20:00] RCPT TO:<recipient@localhost>
[SMTP 000160 31-03-19 23:20:00] 250 Local recipient "recipient@localhost" OK...
```

If you are running Unix or Linux, you can use Postfix, a standard and mature application which will allow you to set up local mailboxes.

EMAIL CLIENT

For Windows, several email applications can render raw messages including Outlook (most users will have this installed with Microsoft Office), mail, and a free App store application called "EML Reader." These should suffice to allow you to open a message and see that it is being parsed properly, especially when checking whether attachments are being encoded and decoded properly.

A test email rendered with Microsoft Outlook:



Sun 3/31/2019 11:20 PM

First on behalf of sender@localhost

Msg Subject

To recipient@localhost

Hello, this is a test message

THE DEBUG OPTION WITH THE SAS EMAIL METHOD

The DEBUG option is an invaluable tool for learning about how the SAS email method generates SMTP. Simply supply DEBUG to your filename statement and then the entire SMTP exchange is written to the SAS log:

```
filename msg email 'recipient@localhost' from='sender@localhost'
sender='sender@localhost' subject='Msg Subject' DEBUG;
data _null_;
  file msg;
  put 'Hello, this is a test message';
run;
R: 220 localhost ArGoSoft Mail Server SMTP Module v.1.0.8.92 at Mon, 01 Apr
2019 04:31:36 GMT
S: HELO localhost
R: 250 Welcome 127.0.0.1, nice to meet you...
S: MAIL FROM:<sender@localhost>
R: 250 Sender "sender@localhost" OK...
S: RCPT TO:<recipient@localhost>
R: 250 Local recipient "recipient@localhost" OK...
S: DATA
R: 354 Enter mail, end with "." on a line by itself
S: Date: 31 Mar 2019 23:31:36 -0500
S: Subject: Msg Subject
S: From: sender@localhost
S: Sender: sender@localhost
S: To:
S: recipient@localhost
S: X-Mailer: 9.04.01M4P110916
```

```
S: MIME-Version: 1.0
S: Content-Type: text/plain;
S:         charset=iso-8859-1
S: Content-Transfer-Encoding: 8bit
S:
S: Hello, this is a test message
S:
S:
S: .
R: 250 Message accepted for delivery (542 bytes,
ID=<tzbil576618ca1y631032019113136@SURFACE>)
S: QUIT
R: 221 Goodbye
Message sent
      To:          "recipient@localhost"
      Subject:     Msg Subject
NOTE: 1 record was written to the file MSG.
```

As you will see when we demonstrate how to send attachments, having this method of examining the SMTP messages that SAS generates will prove very useful.

PUTTING IT ALL TOGETHER

In order to overcome our stated limitations, we will form an entire SMTP message and write it directly to an SMTP server, giving us ultimate control over recipient formatting and all headers. Our code will include the following sections:

1. Save our message headers and body to two separate local files
2. Specify a filename using the socket access method to connect directly to the SMTP server
3. Send the entire message in one connection using a Data step, printing the SMTP conversation to the log

SMTP HEADERS AND BODY

We will start with a very simple example, using data steps to write lines directly to two different local files:

```
filename em_hdr 'c:\temp\headers.txt';
filename em_msg 'c:\temp\message.txt';
data _null_;
  file em_hdr;
  put
    'Sender: sender@localhost' /
    'From: "Last, First" <sender@localhost>' /
    'To: "User, Root " <root@localhost>' /
    'Subject: Test message' /
    'X-Mailer: SAS-Custom' /
    'X-Custom-Header: True'
  ;
run;
data _null_;
  file em_msg;
  put
    'Test message body'
  ;
run;
```

BUILDING AND SENDING THE SMTP

```
filename smtpmail socket '127.0.0.1:25' termstr=crlf;
data _null_;
  length rline sline $255 code 8 fline fname $255 fdata $76;
1 infile smtpmail truncover dlm='00'x;
  file smtpmail;
  link get_response;
  sline = 'HELO LOCALHOST';            link get_send_response;
  sline = 'MAIL FROM: dward@local.me'; link get_send_response;
  sline = 'RCPT TO: root2@localhost';  link get_send_response;
  sline = 'DATA';                      link get_send_response;
  if code^=354 then    2               link smtp_err;

  * Write headers ;
  fid = fopen('em_hdr');
  do while(^fread(fid));
    rc = fget(fid, fline, 255);
    if strip(fline)^='' then put fline;
  end;
  fid = fclose(fid);
  put
    'Content-Type: text/plain; charset=iso-8859-1' /
    'Content-Transfer-Encoding: 8bit' /
  ;                                     3
  * Write message body ;
  fid = fopen('em_msg');
  do while(^fread(fid));
    rc = fget(fid, fline, 255);
    if strip(fline)^='' then put fline;
  end;
  fid = fclose(fid);

  put / '.';    4
  input rline;

  sline = 'QUIT'; link get_send_response;

  stop;
  return;

  get_send_response:
    put sline;
    putlog 'S: ' sline;
  get_response:
    code = .;
    do while(^code);
      input rline;
      putlog 'R: ' rline;
      code = input(scan(rline,1),8.);
    end;
  return;
  smtp_err:
    putlog rline;
    abort;
  return;
run;
```

Notes on the shaded and numbered sections in the preceding code:

1. Notice that the infile and file statements both point to the same filename. The order here is important. This tells SAS to read and write to the same location, in this case the network socket that corresponds to the SMTP server.
2. Once the SMTP server has accepted our sender and recipient(s), it will ask us to send message DATA using the message "354 Enter mail, end with "." on a line by itself" By checking to make sure we receive a 354 before trying to send our message, we are doing some basic error checking.
3. For now we are specifying only plain text as the format of our message. We could also specify text/html if you would like to send a formatted message. This corresponds to the content_type option in the SAS email access method.
4. After sending our headers and message data, we write a period by itself as instructed by the server and by the SMTP specification, then tell the server we are finished by sending QUIT (instead of sending another message).

After running this code, we see something very similar to our previous SMTP messages:

```
R: 220 localhost ArGoSoft Mail Server SMTP Module v.1.0.8.92 at Mon, 01 Apr
2019 05:04:52 GMT
S: HELO LOCALHOST
R: 250 Welcome 127.0.0.1, nice to meet you...
S: MAIL FROM: dward@local.me
R: 250 Sender "dward@local.me" OK...
S: RCPT TO: root2@localhost
R: 250 Local recipient "root2@localhost" OK...
S: DATA
R: 354 Enter mail, end with "." on a line by itself
S: QUIT
R: 221 Goodbye
```

Congratulations! At this point you have sent an email by crafting your own SMTP directly. You can now build additional options, error checking, and add the ability to send attachments, which we will cover in our next step.

ENCODING AND ATTACHMENTS

The SMTP protocol only supports ascii text, so to send attachments we need to do two things: specify a different message content type (rather than text/plain) and encode our attachment. If we first specify a content type of multipart/mixed, SMTP will allow us to specify a message boundary string. This string can be used to separate multiple parts of the message, such as plain text, html, and attachments. We can then go ahead and specify text/plain as the type of our first part, the message body. This would replace the four line put statement at marker 3 of the code on the last page.

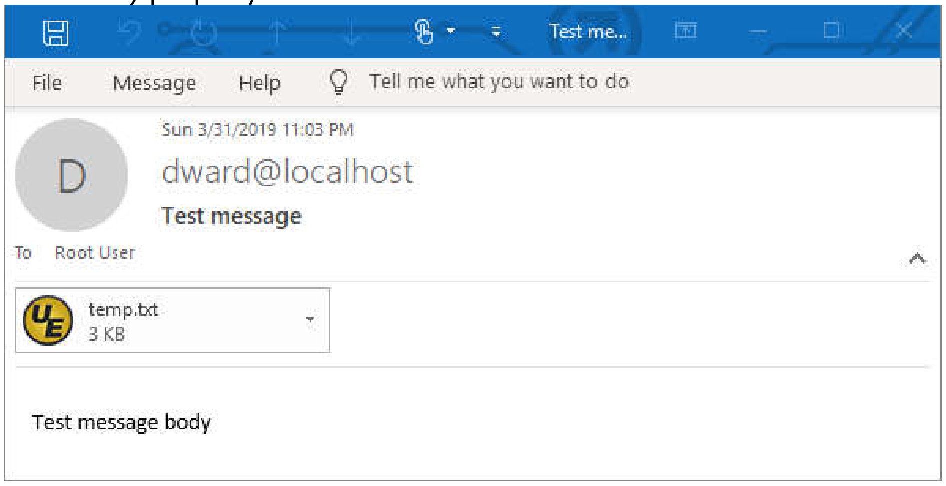
```
put
  'MIME-Version: 1.0' /
  'Content-Type: multipart/mixed; boundary="NEXTPART"' /
  '--NEXTPART' /
  'Content-Type: text/plain; charset=iso-8859-1' /
  'Content-Transfer-Encoding: 8bit' /
;
```

Then, we need to send the attached file, but encoded using base64 encoding. This encoding increases the size of a binary file by 4/3, but translates non-printable characters into a smaller set of characters that is suitable for transmission over SMTP. SAS provides a built-in format $base64x. for encoding ASCII with base64. We will use the macro variable &em_attach to tell our program whether to include an attachment.

```
     if symget('em_attach') ^= '' then do;
       * Write attachment in base64 encoding ;
       fname = scan(symget('em_attach'), -1, '/\');
       put /
         '--NEXTPART' /                                 1
         'Content-Type: application/unknown; name="' fname +(-1) '"' /
        'Content-Transfer-Encoding: base64' /
        'Content-Disposition: attachment; filename="' fname +(-1) '"' /
       ;
       rc = filename('_attch',symget('em_attach'));
       fid = fopen('_attch', 'I', 57, 'B');
       do while(^fread(fid));
         rc = fget(fid, fdata, 57);  2
        fcol = fcol(fid);
        if fcol(fid)<58 then fdata = strip(put(trim(fdata), $base64x104.));
        else fdata = strip(put(fdata, $base64x104.));
         put fdata;
       end;
       fid = fclose(fid);
     end;
     put / '--NEXTPART--' / '.';
```

1. We specify a content-type of application/unknown as a generic solution to force the email client to open the file using an external process.   You can supply other content-types by doing some research and testing whether they function properly.
2. Recall that SMTP has a line length of 78 characters (and two more for the CRLF). Since base64 encodes in multiples of 4, the longest line of encoded data we can transmit is 76 characters, which decoded would be 57 characters of binary data. We open the file in binary mode, read 57 bytes at a time, and for the last “line” if we read less than 57 bytes (by checking the column position using FCOL) we trim trailing spaces.

Our program will now write a multipart message with a body and then an attachment. This screenshot from Outlook shows that it understood both the body and our attachment (and file name) properly:



Sun 3/31/2019 11:03 PM
dward@localhost
Test message
To   Root User
temp.txt
3 KB

Test message body

ENCRYPTION

Over the years, SMTP servers evolved to accept an encrypted socket connection in order to protect the privacy of the messages as they traveled across the Internet or across unsecure networks. With an encrypted connection, the underlying SMTP message remains the same but it is send using SSL or TLS, then decrypted as it is passed to the SMTP server's parser. SAS includes support for encrypted SMTP connections using the emailhost system option:

```
   -emailhost ("smtp.server.com" STARTTLS id=email-user pw=email-pw
  auth=LOGIN port=587);
```

Encryption is typically needed when the message travels across an open network, but most of the time the SMTP server that we connect to will either be on the same computer as SAS, or one within a protected network within which we may not need encryption. Our SMTP server will most likely set up as relayer which connects upstream to another larger and more secure SMTP server such as Microsoft Exchange which will perform lots of verifications and scans to ensure the message does not bounce, that remote servers agree that the recipients are valid, and is free of harmful code or sensitive data. The relayers will be set up to exchange SMTP with other SMTP servers using encryption, so even if the initial connection is not encrypted, as the message changes hands across the internet it will be encrypted.

While it is outside of the scope of this paper, if the SAS to SMTP connection absolutely must be encrypted, the reader should investigate an open source product called Stunnel, or secure tunnel. According to the product description, "Stunnel is a proxy designed to add TLS encryption functionality to existing clients and servers without any changes in the programs' code." SAS would then connect via sockets to a port where stunnel is running, and stunnel would encrypt the data and pass it through to an SMTP server that supports encryption.

CONCLUSION

RECOMMENDATIONS

The code in this paper should be considered an academic exercise. In order to make your own method more complete, you should consider enhancing it in the following ways:

1. Wrapping it in a macro which accepts things like senders, recipients, customer headers, attachments, etc.
2. Doing more error checking before writing the SMTP message – such as verifying email address formats – and when processing responses from the server
3. Supporting multiple attachments

CONCEPTS AND SKILLS

If you have followed our case from requirements to code, you have learned many concepts that have helped you to dive more deeply under the covers of how SAS sends email and hopefully picked up a few new skills that will help you creatively solve problems using SAS code. In summary, here are the concepts and skills presented:

1. What access methods are and how they are used
2. How to use the DEBUG option in the SAS email access method
3. What SMTP is and how to read and write your own messages

4. What network sockets are and how to use the socket access method with SMTP in particular
5. How SMTP encodes attachments using Base64 and how to encode files using SAS
6. How SMTP is encrypted

CONTACT INFORMATION

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

David L. Ward
dward@hdms.com
david@internext.io