Paper 429-2011

The Trilogy on e-Mailing, Part 3: Handling e-Mail Attachments with SAS®

Erik W. Tilanus, Driebergen, the Netherlands

ABSTRACT

This is part three of the trilogy on e-mailing and explains how to handle attachments in incoming e-mails with SAS.

There are two types of attachments: text-based attachments (plain text files, HTML) and binary attachments (anything else; e.g., Microsoft Office documents, pictures). Binary attachments are normally encoded to make them compatible with the text-based character of the e-mail system.

The presentation starts with recognizing attachments in the e-mail data stream. But the main topic is how to decode BASE64-encoded binary attachments and file the resulting binary file.

Part one of the trilogy handled sending e-mails from the DATA step and was presented at SAS Global Forum 2008. Part two handled receiving e-mails in a DATA step; i.e., using the DATA step as a POP3 mail client and was presented at SAS Global Forum 2009.

INTRODUCTION

Part two of the trilogy discussed using the DATA step as a POP3 e-mail client. That presentation described the general process to receive and file e-mails, recognize header and contents parts. During the discussion at the end of the presentation the question was asked how to deal with attachments.

This presentation will try to answer that question.

E-MAIL ATTACHMENTS

Attachments can be of a text nature or a binary nature. Text attachments include HTML files, comma separated files and so on. Binary attachments can be anything: MS Word documents, spreadsheets, picture files and anything else.

Handling an e-mail with attachments is a two step approach. The first step, which is the same for all attachments, is to separate the attachments from the main e-mail body. The second step is handling and filing the attachment itself, The basic handling is similar for both types of attachments, but binary attachments require additional decoding before they can be used.

If the e-mail contains attachments, the mail is considered to be a "multi-part" message. This information is included in the header line "Content-Type: multipart". If you see such header line the next line will be "boundary= "a boundary string" ", which identifies how the parts are separated. There is no standard layout for the boundary string. Normally it consists of a combination of a series of hyphens and a series of randomly selected characters.

Figure 1 shows the record stream you may expect in a multipart message. It starts with all the addressing records and then at some point the boundary definition. Then follow the body of the mail and the attachments, each separated by the boundary line.

So to separate the various parts, you just have to scan for a line that contains the boundary string.

As you can see the lines following the boundary line contains information about the attachment. It defines the type, name and coding method of the attachment.

Text attachments are relatively easy to handle. Just read the records from the mail and file them as the file type which is included in the header lines of the attachment. Binary attachments are more complicated.

SEPARATING AND FILING ATTACHMENTS

Program 1 shows a DATA step that reads and interprets the incoming message. For simplicity sake it has been assumed that the message has been read before and stored in a SAS data set: each line in a separate observation. This process has been described in part two of the trilogy, we will not repeat that here, but simply refer to it.

The program scans the relevant tags in the message to locate and separate the attachments. The program assumes that the attachments have to be stored at some predefined location under their original names. Base64 encoded attachments however are first stored in the SAS temporary (work-)directory. A decoding step will put the restored attachment into the defined location.

When the boundary information ① is encountered it is saved in order to be able to recognize the various parts of the message.

At the beginning of a new message part ② we first examine whether the previous part contained a base64 encoded file. If so, we save the file name and change the file name to "dummy" ③. This closes the previous attachment file. Then we prepare a call to an external program "base64.exe" and call the program to convert the base64 file into its original binary format ④. As an alternative you can write the decoding routine entirely in SAS. This will be discussed in a later paragraph.

Each section of the message contains again several header lines. The "Content-Transfer" line contains the encoding

method ⑤ and the filename= line contains the original name of the attached file ⑥. Just before the actual start of the attachment contents there is an empty line. But empty lines could also be part of the contents of the attachment. Therefore we first test whether we are already reading the body of the attachment ⑦ and if so we link to the output routine. Otherwise the blank line signals the start of the body of the new attachment.

If we are dealing with a base64 attachment, it is written to the SAS Work library ⑦, otherwise it is written directly to the determined destination location.

To write information to the destination files we make use of the FILEVAR option in the FILE statement ⑧. Each time the variable in the FILEVAR option changes its value, the current output file is closed and de-allocated and a new file is opened.

Figure 1: Structure and boundary information of multipart messages

```
Received: from cpsmtpi-eml05.kpnxchange.com ([10.94.77.75]) by CPEXBE-EML16.kpnsp.local
with Microsoft SMTPSVC(6.0.3790.3959);
     Tue, 5 May 2009 21:22:42 +0200
. . .   (more header lines)
Message-ID: <4A00920F.2050102@planet.nl>
Date: Tue, 05 May 2009 21:22:55 +0200From: Synchrona <synchrona@planet.nl>User-Agent:
Thunderbird 2.0.0.21 (Windows/20090302)
MIME-Version: 1.0
To: Erik Tilanus <erik.tilanus@planet.nl>
Subject: testmail with attachment
Content-Type: multipart/mixed;
boundary="------------080808080501060405030609"
Return-Path: synchrona@planet.nl
X-OriginalArrivalTime: 05 May 2009 19:22:41.0490 (UTC) FILETIME=[DBF6DB20:01C9CDB6]

This is a multi-part message in MIME format.
--------------080808080501060405030609      (here starts the body of the mail)
Content-Type: text/plain; charset=ISO-8859-1; format=flowed
Content-Transfer-Encoding: 7bit

There are two attachments to this mail.
The first is a spreadsheet in BASE64 and the second is a text file
Let us get it!
--------------080808080501060405030609      (here starts the first attachment)
Content-Type: application/x-msexcel;
name="test1.xls"
Content-Transfer-Encoding: base64
Content-Disposition: inline;
filename="test1.xls"

0M8R4KGxGuEAAAAAAAAAAAAAAAAAAAAAPgADAP7/CQAGAAAAAAAAAAAAAAAABAAAAHwAAAAAA
AAAAEAAA/v///wAAAAD+////AAAAAB4AAAD////////////////////////////////////
...     (this is the BASE64 encoded contents)
AAAAAAAAAAAAAAAAAAAAAAA4AAAAAEAAAAAAAAAAUARABvAGMAdQBtAGUAbgB0AFMAdQBtAG0A
YQByAHkASQBuAGYAbwByAG0AYQB0AGkAbwBuAAAAAAAAAAAAAAAA4AAIB////////////////
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAFgAAAAAQAAAAAAAAA
--------------080808080501060405030609      (here starts the second attachment)
Content-Type: text/plain;
name="Example data.txt"
Content-Transfer-Encoding: 7bit
Content-Disposition: inline;
filename="Example data.txt"

A,876543
B,Smith,,John,adt,m,19800515,777777555555,john.smith@hulu.com,456456456,137137137,,56
Main Street,,KK11,Princetown,Aruba
C,AUA,BON,3P,100,15MAY2009,Y,1,HK,
D,Sun Tours,20090210,1010
E,Y3M,100,USD,50/10TX/20AP/20VA
F,cc,115,50,USD,,Visa,**********1234,0615,56 Main Street,KK11,Princetown,Aruba
G,4561234567899,1
H,BIKE,,10
I,Check the Credit Card on departure
```

Program 1: Analyze a multipart message

```
      %LET destination_path = e:\testmail\;
      DATA MultiPart;
      LENGTH content_type $30 file_name $255 transfer_encoding $30
             path_file $255  boundary $80;
      RETAIN bodystart 0 content_type " " section_start 0 boundary
             file_name path_file transfer_encoding ;
      SET mailfile;
      SELECT;
①        WHEN (line =: ' boundary=') DO;
            boundary = SCAN(line,2,'"');
            END;
②        WHEN (INDEX(line,TRIM(boundary))) DO;
            IF transfer_encoding = 'BASE64' THEN DO;
             save_path_file = path_file;
③            path_file="%SYSFUNC(GETOPTION(WORK))\dummy";
             FILE atchfile FILEVAR=path_file;
④              cmd= 'c:\base64.exe -d "'||TRIM(save_path_file)||
                    '" "'||"&destination_path"|| TRIM(file_name)||'"';
               CALL SYSTEM(cmd);
            END;
            section_start=1;
            bodystart=0;
            file_name=' ';
           content_type=' ';
           transfer_encoding=' ';
         END;
         WHEN (line =: 'Content-Type:') DO;
            IF section_start THEN content_type=SCAN(line,2,' ');
         END;
         WHEN (line =: 'Content-Transfer') DO;
            IF section_start THEN DO;
⑤             transfer_encoding=UPCASE(SCAN(line,2,' '));
            END;
         END;
         WHEN (line =: ' filename') DO;
⑥          file_name=SCAN(line,2,'="'),2);
         END;
         WHEN (line =: "   ") DO;
⑦          IF bodystart THEN LINK write_rtn;
           ELSE DO;
              IF section_start THEN bodystart=1;
              IF file_name NE ' ' THEN DO;
                 IF transfer_encoding='BASE64' THEN path_file=
⑧                   "%SYSFUNC(GETOPTION(WORK))\"||file_name;
                ELSE path_file="&destination_path"||file_name;
              END;
           END;
         END;
         OTHERWISE DO;
            * these are contents lines;
            IF bodystart = 1 THEN LINK write_rtn;
         END;
      END;
      RETURN;

      write_rtn:
⑧        IF file_name = ' ' THEN OUTPUT;
         ELSE DO;
            FILE atchfile FILEVAR=path_file;
            PUT line;
         END;
      RETURN;
      RUN;
```

BINARY ATTACHMENTS

As mentioned, binary files are more complex to deal with. The SMTP protocol is based on the 7-bit ASCII character set and limits records to 1000 byte, including the terminating CRLF (carriage control-line feed). Binary files by nature are not compliant with that. Therefore these are converted to text files, normally using the "Base64" coding methodology (Figure 2)

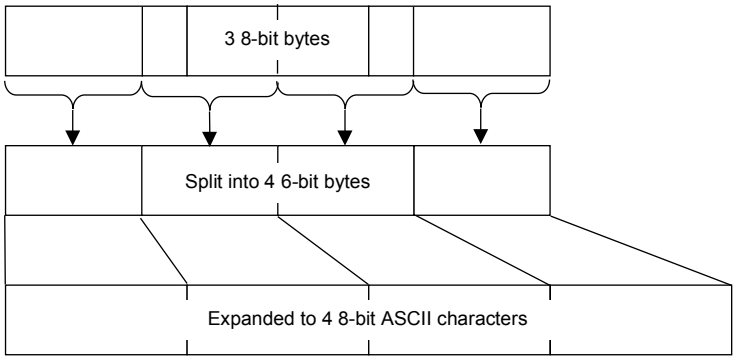
Base64 first expands every three 8-bit bytes into four 6-bit bytes. Next these new bytes are translated into characters using a translation table. If at the end of the input there is no multiple of three bytes left, the base64 encoded file will contain '=' characters in the last one or two positions.

To translate back from the base64 encoded text file you have to do the opposite.

Now comes the caveat. Writing a decoding routine in SAS is not very complicated. So writing out the original binary data to a file should be no problem either, but...

The DATA step FILE and PUT statements are not designed to write binary data. You have to write in a continuous stream, rather than in separate records, which is the normal way for FILE and PUT statements. The key to the solution is to define an extremely large logical record length, large enough to contain the complete decoded file.

Figure 2: The Base64 coding principle



THE CONVERSION ROUTINE

Program 2 shows the basic routine to convert a Base64 file into its original. It reads a line from the input file and then steps through it, getting chunks of 4 bytes at the time ①. Then it inspects whether the chunk contains any "filler" characters ('=') ②. The TRANSLATE function ③ is used to translate back from the characters to the 4 x 6 bit binary codes. If there are one or two filler characters, those bytes are set to '00'X.

Next follows the manipulation to shift the binary information to its proper location in the 3 x 8 bit bytes. The bit manipulation functions require a numeric input. So we first read each byte into a numeric variable wit the IB2. informat ④. From Figure 2 it can be seen that the contents of the first byte just needs to be shifted to the left, to occupy the high order bits in the result ⑤. The two highest significant bits in the second byte should be shifted to the two low order bits in the result. This is achieved by first shifting that second byte 4 bits to the right (i.e. removing the lower 4 bits and move the highest two significant bits to the lower bit positions) and then OR-ing them with the first resulting byte ⑥. In a similar method the other bits are shifted to their right position in the result.

Then we write the resulting one, two or three bytes (depending on the presence of the filler characters) to the output buffer ⑦. When a whole input line has been processed we write the converted bytes to the output file ⑨. Note the trailing @ in the PUT statement. It means that the next PUT will write to the same output record. That is how we achieve a continuous data stream. Only the last PUT statement will actually write out the file in one big chunk. As mentioned in the introduction of this paragraph you need to define a very large logical record length ⑧, larger than the largest attachment you want to handle.

The alternative of writing the decoding routine is to use SAS to write the base64 encoded file to disk as a text file and then use an external base64 conversion program to rebuild the original binary file. An example of such conversion utility is "base64.exe" by Fatih Kodak, which can be downloaded from www.f2ko.de. This is a command-line utility, so you can call it using CALL SYSTEM. This is the method that is already included in Program 1.

THE RESULT

This program has been tested with base64 attachments of over 1Mb of decoded data: spreadsheets, Word documents, PDF files and JPEG files. In all cases the attachments were in their original shape and ready to use.

It is left to the user to integrate Program 1 and Program 2 and combine it with the program to read e-mails from the server.

CONCLUSION

Handling attachments of e-mails is not very complicated. Using the bit-shifting and bit operations functions it is easy to convert a base64 encoded attachment into it's binary original. By defining an extremely large logical record length of the output file and by writing the file at once you can create binary files with SAS.

Program 2: A base64 decoding routine

```
      DATA _NULL_;
      LENGTH line $1000 block $4 decoded $4 binary $256;
      RETAIN binptr 1 binary;
      INFILE 'e:\test.b64' MISSOVER LENGTH=linelength;
      INPUT line $varying. linelength;
①     DO lineptr = 0 TO INT((linelength-1)/4)  ;
         block = SUBSTR(line,lineptr*4+1,4);
②        filler = INDEX(block,'=');
③        decoded = TRANSLATE(block,
            '000102030405060708090A0B0C0D0E0F'x,'ABCDEFGHIJKLMNOP',
            '101112131415161718191A1B1C1D1E1F'x,'QRSTUVWXYZabcdef',
            '202122232425262728292A2B2C2D2E2F'x,'ghijklmnopqrstuv',
            '303132333435363738393A3B3C3D3E3F'x,'wxyz0123456789+/');
         IF filler GT 0 THEN SUBSTR(decoded,filler,5-filler)='0000'x;
④        byte1 = INPUT(SUBSTR(decoded,1,1),IB1.);
         byte2 = INPUT(SUBSTR(decoded,2,1),IB1.);
         byte3 = INPUT(SUBSTR(decoded,3,1),IB1.);
         byte4 = INPUT(SUBSTR(decoded,4,1),IB1.);
⑤        byte1 = BLSHIFT(byte1,2);
⑥        Highorder = BRSHIFT(byte2,4);
         byte1 = BOR(byte1,highorder);
         byte2 = BLSHIFT(byte2,4);
         Highorder = BRSHIFT(byte3,2);
         byte2 = BOR(byte2,highorder);
         byte3 = BLSHIFT(byte3,6);
         byte3 = BOR(byte3,byte4);
⑦        SUBSTR(binary,binptr,1)=PUT(byte1,IB2.); binptr+1;
         IF filler NE 3 THEN DO;
            SUBSTR(binary,binptr,1)=PUT(byte2,IB2.); binptr+1;
         END;
         IF filler EQ 0 THEN DO;
            SUBSTR(binary,binptr,1)=PUT(byte3,IB2.); binptr+1;
         END;
      END;
⑧     FILE 'e:\test.xls' LRECL=1048575;
      binptr=binptr-1;
⑨     PUT binary $VARYING. binptr @;
      binptr=1;
      RUN;
```

WHERE TO GO FROM HERE - RECOMMENDED READING

Part 1 and 2 of the trilogy can be read in the proceedings of previous SAS Global Forums:
Sending E-mail from the DATA step, SAS Global Forum 2008, paper 038-2008
Using the DATA step as a POP3 mail client, SAS Global Forum 2009, paper 002-2008

More on the POP3 protocol can be found in the document on the web site www.faqs.org/rfcs/. Look for the pages RFC 1939, RFC 2449 and RFC 2595.

The bit-shifting functions and the translate function are documented in the Base SAS 9.2 Language Reference.

CONTACT INFORMATION

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

Name	Erik Tilanus	Work Phone:	+31 343 517007
Address	Horstlaan 51	Fax:	+31 84 215 2107
Postal code, City	3971 LB Driebergen	E-mail:	erik.tilanus@planet.nl
Country	the Netherlands	Web:	www.synchrona.nl

At the website you can also find other presentations by the author, held at previous SUGI and SAS Global Forum meetings (including part 1 and part 2 of the Trilogy on e-mailing.