# Let's Not Forget E-Mail

Jack Shoemaker, United Risk Assessment & Management, Cheshire, CT

With all the excitement surrounding the growth and popularity of the World Wide Web, it's easy to forget about that ubiquitous piece of the Internet known as electronic mail, or e-mail for short. Nearly everyone who browses SGML documents - the raw material of the World Wide Web - has an e-mail address to boot. The same can not be said for the converse. Plenty of people have e-mail, but aren't able to browse SGML documents. SGML? Although SGML is the correct way to refer this class of formatted text documents, HTML is more commonly used, and I will simply use the term 'Web document' in this paper to mean anything that you can retrieve with a Web browser.

So, if you're a content provider - someone who creates Web documents in the hope that someone else will drop by for a visit - you've lopped off a good portion of your potential market, if you limit yourself to only Web-based information delivery. And, if you're a potential information consumer with only e-mail access, you have a legitimate gripe with the content provider.

But this is rapidly changing. Although certainly true a year and a half ago, and mostly true even a year ago, Internet access providers have made Web browsing so accessible, that the notion of e-mail without browsing capability is already something of an anachronism. Will the opposite occur? Will there be legions of Web browsers who don't have e-mail? Probably not because e-mail provides an information delivery channel which complements and enhances the Web-based environment.

Consider the friend of many a SAS programmer - SAS-L. SAS-L is a mailing list managed and run by a piece of software known as LISTSERV. Four sites in the world graciously allocate some of their computer resources to keep the LISTSERV program up and running. The program is often referred to as the 'list server'. The list server does three things. It broadcasts mail from one user to all other users on the list; it adds users and their e-mail addresses to the list; and, it deletes users from the list. There's more, but at the core this is all there is to it. Jane has a piece of e-mail she wishes to distribute to everyone else on the SAS-L mailing list. She sends her message along to one of the list servers, that is she addresses the e-mail to say, sas-l@vm.marist.edu, and voila, her message appears in the in mailbox of nearly 10,000 SAS users world-wide.

Contrast this with a Web-based alternative. Jane would write up her message and link it to her home page. She would wait for someone to stop by, notice her link reference, and browse her message. In the first instance, with e-mail, she is pushing the information out to her audience; in the second instance, she is waiting for someone to pull the information from her site.

SAS-L is actually a bit more involved than the preceding discussion lets on. The mailing list and re-distribution function maintained by the four SAS-L peers is mirrored in the USENET newsgroup known as, comp.soft-sys.sas.

Newsgroups are collections of e-mail-like messages organized in such a way as to promote browsing and responding. You need a piece of software called a 'news reader' to read and post 'articles' in USENET newsgroups. In addition you need access to a 'news spooler' which feeds your news reader the news. The popular browsers, Netscape Navigator and Microsoft Internet Explorer, each come with news readers built-in. Most Internet service providers provide a news spool, so in many cases, news reading is available right inside your favorite browser, although some configuration is often required.

Returning to our SAS-L example, the mirroring of SAS-L and comp.soft-sys.sas means that Jane's message is sent not only to everyone on the SAS-L list, but is also posted to comp.soft-sys.sas. Conversely, all postings to comp.soft-sys.sas make their way into the mailboxes of the members of the SAS-L lists. Browsing and responding to newsgroup posts is akin to browsing a Web document. You know where to look. You initiate some browsing. Perhaps something of interest catches your eye. You respond, amplify, or (ahem) correct what you have just read.

As a provider of content, Jane has the best of both worlds. She is pushing her message out to all those e-mail addresses as well as posting her message in a well-known place for others to browse and perhaps even read.

In addition to LISTSERV, there are two other programs in wide distribution which maintain mailing lists: LISTPROC and MAJORDOMO. Each of these programs have their own set of commands, but each allows list owners to establish and maintain lists and allows individuals to subscribe and unsubscribe to these lists - that is have their e-mail addresses added to the distribution lists. In each case, the maintenance, subscription, and use of the list is done by way of e-mail.

SAS-L is an unmoderated, open list which means that anyone can post anything to the list (unmoderated) and anyone can join (open) without approval of the list owner. Not all lists are that wide-open. Consider the Technical Services list operated by SAS Institute. Following the '-L' naming convention of lists maintained by

the LISTSERV software, the Technical Services mailing list is called TSNEWS-L. You can subscribe to this list by sending e-mail to listserv@vm.sas.com which is the address of the SAS Institute list server. In the body of the message place the text: subscribe TSNEWS-L Your Name. The Technical Services department posts information like weather closings and problem alerts on TSNEWS-L, so if you are a subscriber, you'll receive periodic messages from SAS Institute about weather closings and problem alerts. SAS Institute allows anyone to subscribe to this service, so the list is open. Posting to the list is another matter. Only approved Technical Services announcements are allowed on TSNEWS-L, so the list is moderated. Which means, in effect that you or I can not simply post a message to TSNEWS-L.

Consider the benefit of TSNEWS-L as a subscriber. The problem alerts are also available on the SAS FTP server and the SAS Web Server. So you could locate the URL of these alerts and periodically check for new alerts. Why waste the time? If you know that you would like to hear about a problem alert as soon as possible, a subscription to TSNEWS-L will guarantee that a new alert will show up in your mailbox as soon as it is released. SAS Institute places the problem alerts out on its servers so that you can pull the information at your convenience; however, SAS Institute also provides a mechanism to push these alerts to you should you wish. SAS Institute also has a list called NEWDOCS-L which sends out notices about new books and documentation. This is another example of information which may be pulled from the appropriate SAS server, or pushed to you by way of a LISTSERV mailing list.

Problem alerts aren't the only type of information that you, as a company, would like to push out to your customers. New product offerings, announcements, and general marketing material make excellent sources of content for company-based mailing lists. The problem is getting people to subscribe. New product offerings, announcements, and general marketing material also make excellent sources of content for Web documents. In fact, cynics have remarked that this is all the World Wide Web is: nothing more than a fancy, expensive, and slow version of the Yellow Pages provided by your local telephone operating company. True enough, up to a point. The scope of the World Wide Web is, well, world-wide. Furthermore, well-designed Web sites are rich in useful content, not just full of doo-dads and other clutter.

Because e-mail has existed since the birth of the Internet - well almost since the birth - there are plenty of mailing lists available for subscription. But where do you find them? Not all mailing lists use the LISTSERV software, so you can't rely on 'listserv@' working all the time. In contrast, the World Wide Web has been in existence for a scant five years (tops!), yet finding a home page is normally a snap because the convention is to establish a host called 'www' which handles incoming Web requests. In other words, if you'd like to find out something about the company DotDot , using the URL http://www.dotdot.com will probably work if DotDot has established a Web site. No such convention exists for the e-mail addresses of mail robots like the LISTSERV program. But it should.

At a minimum, DotDot should establish valid e-mail addresses for 'info' and 'help'. One address may be the alias of the other. That is, unless there is a desire to handle requests to 'info' differently than 'help', there is no need to establish two separate responders. You'd like to have both addresses available externally so that as many people as possible can contact you. In any event, 'info', or 'help' can respond with an automatic response that points the requester in the right direction. To the DotDot list server, for example. Other useful addressees include '411', '611', '911', and 'sales'. Would that every company that supports e-mail have at least an 'info' and 'help' e-mail address. '611' - repair, and '911' - emergency are best suited for internal requests. That is, they are wonderful complements for a Web-based intranet.

The e-mail administrator at your site can establish accounts for 'info' and 'help' as described above. Depending on your mail server software, you may be able to set up these accounts to respond unconditionally with a pre-defined message. Alternatively, these addresses may be forwarded to an individual, or individuals, responsible for responding to the content of the messages. For a small site, this might be fine. As the volumes of these messages increase, you will certainly want to explore automated means for responding to requests which come in by way of the 'info', or 'help' accounts.

You can also use e-mail to distribute data and documents. Some material is not suited for e-mail delivery: a picture of Saturn or some other heavenly body, or a point-and-click map of your state with links to county-based information. When all the pointing and clicking is done; however, many Web-based requests terminate with information displayed as a chunk of text or table of numbers. These type of data can use e-mail as the information delivery channel as well.

Consider the SAS Notes server maintained by SAS Institute. You can get to the Notes server using your favorite browser, or you can use e-mail to interrogate the server directly - first for a list of potential notes; then for the actual notes themselves. With either method your final objective is the SAS note - a message from SAS Institute describing a perceived or actual problem with a recommendation for action and sometimes a promise to fix the problem in the near future with a future release of the SAS system.

In the previous example, you are using e-mail to pull information from the SAS Notes server. Whether you use e-mail or Web-browsing as the delivery channel is a matter of personal preference and technical infrastructure. If you have no browser, or if you must browse over a connection which is something less than one megabit, then you will thank SAS Institute for providing an e-mail

entrance into this information. If you have a browser at your desktop with a high-grade, fast connection, you may choose to click, click, click. Just be sure to 'bookmark' the actual page where you can fill out the form which is sent to the server to satisfy your request.

The e-mail version of the SAS Notes server works because there is a program attached to the other end of the e-mail request which parses the message body and carries out the specified actions. This requires something a bit more sophisticated than the auto-responder suggested for 'info' and 'help'. Many mail readers have functions with names like 'Rules' or 'Filters' which allow you to provide e-mail file servers with little effort.

For example, suppose you have a small number of files which you wish to distribute on demand called file1 - file5. If your mail reader supports rules, or filters, you may be able to set up a rule which goes something like this:

> if the Subject: field contains GET FILE1
> then Send FILE1 to the From: address

The actual syntax of the rules varies from one piece of software to the next. Also, if you are using a Windows-based mail reader you will probably have to set these rules up in a series of dialog boxes instead of simply typing some rules into a configuration file. Another drawback to this approach is that you need a separate rule for each file. For five files, this is manageable; for one hundred it is not.

You can create a more flexible solution if your mail reader has a rules syntax that allows some sort of substitution. For example, we could generalize our example above if the we could test for the existence of the word 'GET' at the beginning of the Subject: field, then use the second word as the name of the file to send. Under UNIX, you can use a program like ProcMail to achieve this flexibility. Briefly stated, you would forward mail to the ProcMail program which then uses a ProcMail script that you have written. The ProcMail scripting language allows you to examine any, or all, of the header fields, and the message body. You can use regular expressions to match any character pattern you wish. And, based on the results of these matches, ProcMail can either hand over the text to another process (piping), send back a response, or do both. For all practical purposes you can make e-mail do anything you want when used in conjunction with a mail filter like ProcMail.

Another benefit of this approach is that you can use any e-mail account, including your own, to act as a mail robot. This is particularly useful if you can't persuade your e-mail administrator to establish and maintain some mail robots for you. The disadvantage is that ProcMail runs under UNIX and therefore can not help you if you don't have UNIX platforms at your site handling mail distribution. Fortunately, as a SAS user you have another way out.

With release 6.11 of the SAS system, you can now use the SOCKET access method in conjunction with the FILENAME statement to read and write directly to TCP/IP ports. (Note: The new 6.11 access methods CATALOG, FTP, and SOCKET are documented on pages 3 - 14 of SAS document 55300, "SAS Software: Changes and Enhancements, Release 6.11".) This access method allows you to create, in effect, an e-mail reader using SAS because e-mail normally arrives by way of a TCP/IP service known as a 'POP3' server and is sent out by way of an 'SMTP' receiver which is also a TCP/IP service.

The POP3 server is nothing more than a program which runs continually and polls, or listens, to a specified TCP/IP port - usually 110. That is, port 110 is the "well-known" port assignment for the POP3 server. The SMTP server, on the other hand, does it work on port 25.

The POP3 server accepts a small set of commands defined in a document called RFC 1725. These commands are of particular interest: USER, PASS, LIST, and RETR. The first two commands are used to log into your mail account. The third command lists all the messages in your mailbox. The last command retrieves a particular message. No matter the display of your favorite mail reader, remove the window dressing and you'll find that these simple commands are being sent to port 110 on the host which is running the POP3 server.

Here's an example of using SAS to interrogate a POP3 server at our company called DotDot. The POP3 server is a program called IMS which runs on the host known as 'mailhost'. First, establish a channel to the POP3 server using the SOCKET access method in a FILENAME statement.

```
filename pop3  socket 'mailhost:110';
```

In a subsequent data step we will use pop3 as the filename for a FILE statement to write commands and an INFILE statement to read responses. The keyword, SOCKET, indicates that this filename statement uses the SOCKET access method. Inside the quotes is the name of the host followed by a colon followed by the desired TCP/IP port, in this case 110.

We begin our interrogation with an INFILE statement to open the channel for subsequent INPUT statements.

```
data _null_;
  infile pop3 length = len;
  input;  file log;  put _infile_;
```

The input statement is necessary to read the welcome message from the POP3 server. The 'file log' and 'put _infile_' statements are not necessary; however, they allow you to see what's going on. Here's what the SAS log looks like:

```
+OK IMS POP3 Server 0.87 Ready
```

There will be some more information to the right of 'Ready'. I have truncated it for the purpose of clarity. Also note that if you are not running IMS version 0.87, you won't get exactly the same message. In any event the POP3 server is ready to accept some commands. First we need to log into out mail account using the USER and PASS commands as follows:

```
file pop3;  put 'USER shoe';
input;  file log;  put _infile_;

file pop3;  put 'PASS mypass';
input;  file log;  put _infile_;
```

Again the 'file log' and 'put _infile_' statements are included to display the results on the SAS log. You could re-direct these elsewhere, or eliminate them all together. What is important is to code an 'input' statement after a 'file'-'put' pair. Also, you need to supply a valid user name and password. In this example I have used the user name 'shoe' with the password 'mypass'. Here is what the SAS log looks like for these statements.

```
+OK shoe is welcome here
+OK shoe's mailbox has 4 message(s)
```

These reassuring messages mean that I have an account at 'mailhost' and that there are four messages in my mailbox. If I want to read all the messages - one-by-one, I need to know how many messages there are to retrieve. We could do this by parsing the string returned after the PASS command, but let's see what the LIST command does.

```
file pop3;  put 'LIST';
```

The LIST command instructs the POP3 server to list out each message in the form message-number, message-id, where message-number is a sequential number starting at 1 and message-id is an internal message ID generated by the POP3 server. When the server has exhausted the list of messages, it writes out a single dot ('.') which means we are at the end of the list. This makes our input statement a bit more complex because we need to loop through input statements until we reach a dot. Also, we don't want to write any more commands to the server until we have consumed the whole list. Consider this code fragment:

```
do until ( a = '.' );
  n = a;
  input a $char1.;
  file log;  put _infile_;
end;
```

The 'do until' loop will execute at least once and will continue looping until a '.' is encountered as the first character of the input buffer. The input statement reads the first character of the input buffer into a data step variable called 'a'. When the first character of the input buffer is '.', a will have a value of '.' and the 'do until' condition will be true so the loop will stop executing. When the first character is not '.', it is the message number. In a real-life example, we would account for message numbers greater than 9; however for this exercise it will suffice to read only the first character of the input buffer in order to retrieve the message number. When the loop has finished executing we would like the data step variable n to contain the last message number. That is, we would like n to be the last value of a before the terminating '.'. Placing the 'n = a' assignment before the input statement accomplishes this nicely. The first time through the loop, a has a value of missing, so n does as well. After the 'n = a' assignment, the first character of the input buffer is read into the data step variable, a. The 'file log' and 'put' statements are placed inside the loop for the purpose of displaying the response from the POP3 server. The program will work fine if they are removed. If a is not '.' then the loop iterates. The first action is the 'n = a' assignment. So on the second pass through the loop, n has the value of the value of a from the first pass. The process continues until the terminating '.' is encountered. When this happens, the loop stops executing and n is left with value of the next-to-last value of a - exactly what we wanted. Here are the log statements from this segment:

```
+OK 4 messages (1625 octets)
1 404
2 407
3 407
4 407
.
```

At this point the data step variable n will have a value of 4 because there are four messages in the shoe mailbox. To read each message, in turn, we issue a 'RETR' command. For example:

```
do i = 1 to n;
  file pop3;  put 'RETR ' i 2.;
  .. code to read mail contents ..
end;
```

The do loop will iterate from 1 to 4 and retrieve messages 1 through 4. Both the mail headers as well as the message body come back from the POP3 server. The chunk of code which is omitted in the above code fragment is the stuff which actually does the work on the headers and message body. It is the section of code which parses the mail message and takes appropriate action. That action can be anything that a SAS program can do. Furthermore, you have the full power of the SAS system at your disposal to parse the mail message. All-in-all it's a rather powerful cocktail of a flexible, world-wide information delivery channel and robust data manipulation environment.

To demonstrate how you can use SAS to manipulate the mail message data, consider the code fragment below. The code examines the header portion of the mail message and writes the 'Subject:' and 'From:' records to the SAS log. Mail headers have the form field-name, colon, field contents. So the subject field is the record which begins with 'Subject:'. The remaining portion of the record is the contents of the 'Subject' field.

```
do until ( a = '.' );
  input @1 a $char1. @;
  input @1 record $varying200. len;
  header = scan( record, 1, ':' );
  if header in ( 'From', 'Subject' )
    then do;
    file log; put _infile_;
  end;
end;
```

Each mail message terminates with a '.' in the first position of the input buffer. So we re-use the 'do until' strategy used to read the results of the 'LIST' command. Because we want to examine the contents of each record, rather than just write the contents to the SAS log, we will read the input record into a data set variable called record. Since the input records have varying length, we need to use the $VARYING informat which requires that you supply a data set variable which contains the desired length of the data step variable, record. On our first INFILE statement, we used the LENGTH= option to establish a data step variable, len, which will contain the record length of the current input buffer. This value is set by the INPUT statement. So, the first INPUT statement inside the 'do until' loop performs two tasks. First, it reads the first character of the input buffer into the data step variable a. Second it places the length of the current record in the data step variable len. The '@' at the end of the first INPUT statement holds the column pointer so we can re-read column one and the entire record using the data step variable len to set the length of record.

Next we use the SCAN() function to examine the first colon-delimited word in each record. If the first word is either 'From', or 'Subject', we write the entire contents of the input buffer to the SAS log. Of course this is only an example. For a real-life mail robot you would do much, much more with this information. In any event, here is the SAS log for this code fragment.

```
From: Jack N Shoemaker <shoe@std.com>
Subject: Test message 1

From: Jack N Shoemaker <shoe@std.com>
Subject: Test message 2

From: Jack N Shoemaker <shoe@std.com>
Subject: Test message 3

From: Jack N Shoemaker <shoe@std.com>
Subject: Test message 4
```

Finally, to terminate the POP3 session, issue the 'QUIT' command and stop the data step from executing again.

```
file pop3;  put 'QUIT';
stop;
run;
```

The _null_ data step executes just once because all reading and writing to the POP3 server is handled inside the data step. Here's the NOTE: statement corresponding to the INFILE and FILE statements:

```
NOTE: The file/infile POP3 is:
      Local Host Name=WNT40,
      Local Host IP addr=205.206.99.100,
      Peer Hostname Name=
        mailhost.dotdot.com,
      Peer IP addr=205.206.99.97,
        Peer Name=N/A,
      Peer Portno=110,
        Lrecl=256,Recfm=Variable
```

The 'Local Host Name' and 'Local Host IP' refer to the machine running this SAS program. The 'Peer' references are to the machine which is running the POP3 server. In this case the POP3 server is running on a host called 'mailhost.dotdot.com' on port 110. Stripped of the 'file log' statements, here is the bare-bones SAS code:

```
filename pop3  socket 'mailhost:110';

data _null_;
  infile pop3 length = len;
  input;
  file pop3;  put 'USER shoe'; input;
  file pop3;  put 'PASS mypass'; input;
  file pop3;  put 'LIST';
  do until ( a = '.' );
    n = a;
    input a $char1.;
  end;
  do i = 1 to n;
     file pop3;  put 'RETR ' i 2.;
     do until ( a = '.' );
       input @1 a $char1. @;
       input @1 record $varying200. len;
       header = scan( record, 1, ':' );
       if header in ( 'From', 'Subject')
         then do;
           file log; put _infile_;
       end;
```

```
    end;
  end;
  file pop3;  put 'QUIT';
  stop;
  run;
```

Say you wanted to send out a file called c:\SUGI\Example.File. Using a similar technique, you can use SAS to communicate with the SMTP receiver on port 25. Like the POP3 server, the SMTP server understands a small set of commands described in RFC 821 and RFC 1123. Of particular interest are HELO, MAIL, RCPT, and DATA. First we need to establish FILENAME references for the SMTP receiver and the file which we wish to send.

```
filename smtp  socket 'mailhost:25';
filename response
  'c:\SUGI\Example.File';
```

We start our conversation with the SMTP receiver by identifying where we are, who we are, and to whom we would like to send the mail. The HELO, MAIL, and RCPT commands, respectively, do this.

```
data _null_;
  infile smtp;

  input;
  file log; put 'INIT>' _infile_;

  file smtp;
  put 'HELO dotdot.com';
  input;
  file log; put 'HELO>' _infile_;

  file smtp;
  put 'MAIL FROM:<you@dotdot.com>';
  input;
  file log; put 'FROM>' _infile_;

  file smtp;
  put 'RCPT TO:<shoe@std.com>';
  input;
  file log;  put 'RCPT>' _infile_;
```

The 'file log' and 'put' statements are not necessary; however, this is what was written to the SAS log:

```
INIT>220 mailhost.dotdot.com

HELO>250 OK

FROM>250 shoe@dotdot.com OK

RCPT>250 shoe@std.com OK
```

At this point you are ready to send the body of the mail message. In this case the body of the mail message will be the file called c:\SUGI\Example.File. The 'DATA' command tells the SMTP receiver to start receiving the message body.

```
file smtp; put 'DATA'; input;
file log; put 'DATA>'_infile_;
```

The SMTP receiver responds:

```
DATA>354 Ready for data
```

The following code segment writes the contents of the file to the SMTP channel and therefore into the message body of the mail. When the input file is exhausted, a '.' is sent to SMTP to indicate the end of the message body.

```
file smtp;

do until ( lastrec );
   infile response end = lastrec;
   input; put _infile_;
end;

put '.'; input;
```

Finally, the SMTP session is completed with the 'QUIT' command and the mail message is sent along for delivery.

```
put 'QUIT'; input;
```

The SOCKET access method available with release 6.11 allows a SAS program to communicate directly with the TCP/IP ports by way of the FILENAME statement. Data may be read and written to and from these ports directly using the SOCKET access method and a simple data _null_ step. Since TCP/IP is the default protocol for most inter-network information exchange, the SOCKET access places the entire power of the SAS system at the disposal of any TCP/IP-based inter-network information exchange application. And that includes the old work horse of the Internet: e-mail.

The author welcomes comments and criticisms.

Jack N Shoemaker
United Risk Assessment & Management
Box 74, 288 Highland Avenue
Cheshire, CT 06410

203 250 8618  shoe@world.std.com