

Confessions of a SAS® System Electronic Mail Junkie

Derek Morgan, Washington University Medical School, St. Louis, MO

Abstract

My first use of the SAS System electronic mail interface was for the 1999 International Genetic Epidemiology Society meeting. To reduce administrative costs, the organizers wanted to use electronic mail as the primary approach for confirming registration and abstract submission, and the e-mail interface performed beautifully.

Suddenly, I find it in many other things I am doing, several of which weren't done by e-mail before. I use e-mail to provide rapid feedback on process quality control, and for communication with remote users of our in-house SAS System application. This presentation will detail those other uses, and show a simple SAS/AF program module which drives the communication for remote users. Anyone who uses e-mail to distribute results, perform process notifications, and has not discovered the wonders of the SAS System e-mail interface will benefit.

Review #1: IGES '99

The SAS System was selected for registration and abstract management based on the existence of similar in-house software developed for another conference. Registration confirmation in the original software was handled via printed letters mailed to registrants. The regional nature of the original conference allowed for accurate budgeting of postage costs. However, IGES projected many attendees and presenters from overseas, adding airmail costs, the special handling of overseas envelopes, and extra aggravation for the conference coordinator.

Electronic mail was clearly advantageous in terms of cost, but how would it fare in the convenience category? The administrative applications were ported to our UNIX server. With its direct connection to our mail server there were no obstacles to communication across machines. It was also completely transparent to any user, with the only possible indication of an electronic mail link being the email address field on the data entry screen.

The ability of the SAS System electronic mail interface to read the recipient's address from a column in a table made it easy to generate a customized electronic letter in a DATA step that contained the registration number, payment status, events registered for, and standard text with general conference information. The subject line also used values from the registration or abstract table, and it sent copies to the appropriate people. A second e-mail report was also generated after the edit session containing a summary of the transactions for the session.

The front end of the application is a custom FSEDIT screen invoked from a UNIX command line with a single word (through aliasing). The software is accessed using an X terminal running on Windows NT clients, allowing use of the software from any machine with the correct permission. Although there were separate commands for

registration and abstract management, it would have been a simple matter to tie everything together using SAS/AF® to create an integrated, menu-driven conference management system. Fortunately, the conference came and went, and nobody asked for that.

Review #2: Using The SAS System Electronic Mail Interface

Adding electronic mail capability to SAS System applications is a matter of creating the text of your mail and using the e-mail tokens to address, give the message a subject, and send the mail. There is also a token to send a copy of the mail to other recipients. The easiest way is to create the text of the mail using PUT statements inside of a DATA _NULL_ step. It's old technology, but easy to modify and debug. This method also has the advantage of being separate from the editing of the tables from which it obtains information.

You need to start the SAS System with the -EMAILSYS system option. This cannot be set inside the program, but must be defined at invocation of the session. What the parameter for the option will be is operating system-dependent. For UNIX, there are two possibilities: one that allows for aliasing and attachments, and one that does not.

Once it has been started, define a FILENAME with the EMAIL option as below:

```
FILENAME mailout EMAIL 'derek@wubios.wustl.edu';
```

This sets up the file reference where the output is to be directed and gives it a default address. There are multiple ways to specify the actual address to be used for the mail. We used the e-mail token "!EM_TO!" inside the DATA step that created the text of the mail. In this DATA step, first we direct the output to the file reference we specified earlier:

```
FILE mailout; /* MAKE SURE YOU AREN'T SENDING  
THIS TO AN ODS DESTINATION! */
```

Next, we address our mail, give it a subject, and specify the address where a copy of the mail is to be sent.

```
PUT '!EM_TO!' email;  
PUT '!EM_SUBJECT! Registration Confirmation #'  
    regno z3.;  
PUT '!EM_CC! derek@wubios.wustl.edu';
```

The address is contained in the table variable "EMAIL". Note that the e-mail token and fixed text for the subject line are within the same set of quotes. A registration number ("REGNO") is attached to the subject line. Since the CC: address doesn't change with each record, it is hard-coded in the "!EM_CC!" token.

After the mail address and subject have been set up, PUT statements provide the means for the e-mail text. The DATA step allows for customization based on table values as demonstrated by the following example:

```
PUT 'You have registered for:';
IF regtype GT 0 THEN
  PUT 'IGES Conference: ' regtype regtyp.;
IF morton THEN
  PUT 'Morton Symposium';
IF banquet1 GT 0 THEN
  PUT @1 banquet1 2. 'extra banquet tickets';
```

In this example, each item is on a separate line. Use of a trailing "@" with the PUT statement would allow for the "flowing" of text.

After the statements creating the text for the message have been coded, use the token "!EM_SEND!" to actually send the message at the end of the record. Without this token, only one message would be sent using the data from the last record in the table. Following that, you need to clear the previous message and create a new one for sending to the next person in the table. The "!EM_NEWMSG!" token does that, clearing the recipient and copy to lists, the subject line, and text of the message. A special token, "!EM_ABORT!" is used to override the default behavior of the electronic mail interface. Normally, the message is automatically sent at the end of the DATA step. Using this token will prevent that from happening when there are no more records to be processed. Otherwise, a blank final message (because of the EM_NEWMSG token) would be transmitted to any addressees that are hard coded in the TO and CC tokens. The EM_ABORT token must be used in conjunction with the EOF option on the SET statement.

```
/* Send message and clear for next message */
DATA _NULL_;
SET maillist EOF=last; ①
...
PUT '!EM_SEND!';
PUT '!EM_NEWMSG!';
RETURN;

/* Don't send at end of DATA step */
last: PUT '!EM_ABORT!';
RUN;
```

The EOF option on the SET statement (①) is necessary for the proper operation of the EM_ABORT token. Only the very last send operation should be aborted, which occurs during the last record. Therefore, the EM_ABORT token should only be executed during the last record. This prevents the default send operation from occurring.

Debugging the e-mail text and/or the process is usually simple, most frequently involving a missing exclamation point for one of the tokens, or that perennial SAS System favorite, mismatched quotes.

As mentioned earlier, a second e-mail report is generated using a separate DATA step. Instead of one e-mail per record, however, we want one e-mail for the whole table. In this case, we can dispense with the "!EM_SEND!", "!EM_NEWMSG!", and "!EM_ABORT!" tokens, relying on the default of sending the e-mail message at the end of the DATA step.

```
DATA _NULL_;
SET tosend;
BY DESCENDING econfirm;
FILE mailout NOPRINT;
today = TODAY();
PUT '!EM_TO! derek@wubios.wustl.edu';
PUT '!EM_CC! techprog@wubios.wustl.edu';
PUT '!EM_SUBJECT! Conference Confirmation
      Summary ' today date9.;
IF first.econfirm THEN DO;
  PUT ' ';
  PUT "The following registrations were " econfrm
      econfrm.;
  PUT "Reg #" @8 "Name";
END;
PUT @1 regno z3. @8 name;
RUN;
```

With the exception of the address, copy, and subject lines, this is nothing more than a simple DATA step report. This is why adding e-mail capacity is easy. My in-house client, and my supervisor, and I were all happy with the result. That should have been the extent of my SAS e-mail efforts (I used a clone of the process for DiamondSUG registration.)

New Horizons

I am responsible for the development and maintenance of PC-based data management software for medical studies. Each time a new phase begins, I update the data management system software, which is SAS System based, using SAS/AF and SAS/FSP. Our collaborators also get new computers at each phase to try to keep up with the hardware curve. We install the software and prep each machine on-site before shipping them it to our collaborators' field centers.

For the newest phase of our study, each machine was equipped with an Ethernet card, since all of the field centers now had Ethernet access. This engendered a bright idea: we could transmit data electronically rather than have it copied onto portable media and sent via Federal Express. With SSH (secure socket) transmission for security, we could decrease the cost of data transfer dramatically! That was bright idea #1. However, the next question to be answered was how would each field center notify us that they had made a data transmission? The answer: by E-mail, of course.

That's when I had another great idea. Instead of making the person doing the transmission notify me when they had sent data, why not have the data management system itself notify me? Since I was starting SSH from within the application with a CALL SYSTEM, and we had access to a MAPI32-compatible PC-based e-mail system, what was to stop me from using the SAS System E-mail interface to let me know when the CALL SYSTEM was executed? Nothing.

Using Electronic Mail via SAS Component Language

In SCL, FPUT and FWRITE are the commands that send text to an external file. A default filename still has to be created, and the tokens still have to be used in the same manner, but otherwise, it's exactly the same process. Who should receive this notification was be hardcoded, since the personnel responsible for this study was already

known. There is a SAS System table on each machine with one record that contains machine-specific registration information. This table happens to have the name of the field center as one of the columns, which allows us to know who sent the data. We could even put all of the necessary information in the subject line: "Data sent from xxxxxx". My supervisor was happy, our collaborators were happy... However, I wasn't quite content to stop there. What else could I do with this e-mail capability?

Well, how about having the software notify me if something doesn't go right? That's something else I could do! And what about the system patches that require running a SAS System program? I could have it send me the log when they patch the system. I can also see who is timely about applying the updates. Suddenly, this phase of the study became filled with e-mail possibilities. I could notify the field centers about problems with their data and/or their technicians from a centralized application running on our UNIX server when I combined the data every week! I could even... wait. I have an alpha-test deadline three weeks away. The question then became what was it practical to do in that time frame, and how to code it.

Getting An Application to Tell You What It's Doing via E-mail: DISTRESS.PROGRAM

The following SCL code is how all e-mail is sent from the application. By passing parameters to the SCL module, it can handle sending the transmission notification, the software patch notification, and it can also tell you when things aren't quite working right with the application itself (the latter two with the SAS System log attached).

```
/* DISTRESS.SCL – Send e-mail to Coordinating
Center From Data Entry System */

/* Get title and log file name from calling module */
ENTRY title $ 80 logfilename $ 80; ①
INIT:
/* Set up recipients */
prog = "derek@wubios.wustl.edu";
copy = "notreal1@wubios.wustl.edu,
       notreal2@wubios.wustl.edu";
LENGTH site_id $ 10;
disptitl = title; /* Set title field in display window */

/* Set reason field in display window */
if INDEX(title,"Snapshot sent") THEN DO;
    errorkey = 0;
    reason = "the snapshot has been transmitted.";
END;
ELSE DO;
    errorkey = 1;
    reason = "there is a problem.";
END;
REFRESH; /* Refresh display window */
rc = rc;
```

```
/* Get Field Center Information from registration table */
dsn = "hgx.register (READ=" || SYMGET('rdpswd') || ")";
register = OPEN(dsn,'I');
CALL SET(register);
rc = FETCH(register);
rc = CLOSE(register);

MAIN:
/* Assign logical filename to EMAIL device */
rc = FILENAME('HELPME',prog,'EMAIL');

/* Open output file for e-mail creation */
mailfile = FOPEN('HELPME','A');
IF mailfile THEN DO;
    rc = FPUT(mailfile,"!EM_NEWMSG!");
    rc = FWRITE(mailfile);
    rc = FPUT(mailfile,"!EM_TO!"||prog);
    rc = FWRITE(mailfile);
    rc = FPUT(mailfile,"!EM_SUBJECT!"||title);
    rc = FWRITE(mailfile);

    IF SYMGET('userid') NE 'WUDCC' AND
        SYMGET('userid') NE '' THEN DO; ②
        rc = FPUT(mailfile,"!EM_CC!"||copy);
        rc = FWRITE(mailfile);
    END;
    IF logfilename EQ '' THEN DO;
        /* Close log file in progress */
        SUBMIT CONTINUE;
        PROC PRINTTO; ③
        RUN;
        ENDSUBMIT;
        logfilename = SYMGET('rootlib') ||
            "\tmp\hgendes.log";
    END;
    /* If it's an error, attach the log */
    IF errorkey THEN DO;
        CALL WAIT(2);
        rc = FPUT(mailfile,"!EM_ATTACH!"||logfilename);
        rc = FWRITE(mailfile);
    END;
    ELSE DO; /* No problem, just data */
        mailtext = site_id || " has transmitted their
            snapshot.";
        rc = FPUT(mailfile,mailtext);
        rc = FWRITE(mailfile);
    END;
    rc = FCLOSE(mailfile);
END;
ELSE /* Print message if mail cannot be sent */
    _msg_ = SYSMSG();
rc = FILENAME('HELPME',' '); /*clear E-mail filename */
CALL WAIT(5);

TERM:
SUBMIT CONTINUE; /* Resume logging */
PROC PRINTTO LOG=LOGFILE;
RUN;
ENDSUBMIT;
CALL EXECCMD('CANCEL'); /*Close Window */
RETURN;
```

An Explanation Would be Nice

This module is designed to send e-mail in a variety of situations, and can include the session log file or an external log file as an attachment. The module is invoked

by the SCL command CALL DISPLAY (remember, this is a SAS/AF application, not a FRAME. However, the SCL should be suitable for use in a FRAME.) The title of the e-mail to be sent and the name of an external log file to attach are sent as parameters (①). The log file name is left blank if the session log is to be attached. Since the recipients of the mail being generated are known, they are hardcoded here. Alternately, a recipient list could be generated from a data table. The display window (fig. 1) has a title and a reason for the e-mail. This is all that the user sees. The field center information is obtained from the machine registration table, so that the recipient knows who sent the mail.

Figure 1: Display Window

```

      Invoking automated distress call system...

The system has encountered a problem.

      This e-mail will be sent with the title:
Error Updating FORMSTAT

      Please wait... this window should close automatically...

      If it does not close in 10 seconds, press F8 to close it manually.

```

MAIN is where the e-mail is assembled using FPUT and FWRITE statements and the e-mail tokens. The !EM_ATTACH! mail token is executed only if a log is to be attached to the mail. A line of text is written for data transmission. Item ② prevents the "Cc:" field from being used, so that the rest of the team doesn't get e-mail during debugging. The session log is closed for attaching in ③ with PROC PRINTTO. You can not pull the session log from the log window, so redirection is necessary when the data management application is started (A PROC PRINTTO is executed in the autoexec.sas file.) The TERM section resumes logging in the session log file, and closes the display window.

This application runs on the Windows platform using Eudora 5 as the MAPI32-capable mail software. DISTRESS.PROGRAM was my first SAS System e-mail "fix" during this project. The next stage was providing rapid feedback to field centers for quality control of exams.

This part didn't involve much new programming, but it was another use for the SAS System E-mail interface. Part of the data transmitted involves ASCII files generated by a separate application at the field center. Each record in the file has a key identifier describing the type of measurement, and when it was taken. By reviewing these files, we can determine if the person administering the exam followed the correct protocol. If not, we need to explain what was done incorrectly.

An FSEDIT screen application was built on our UNIX server where the aggregate data is assembled and stored. The table referenced by this screen contains fields for Subject Identifier, the date of the exam, the technician administering the exam, and a comment. If there is a problem during the exam as evidenced by the

data, the person reviewing the exam will add a record to the table explaining the problem with a particular exam.

When all of these comments have been entered (and as a part of the aggregation process), E-mail is generated (using DATA _NULL_) for each of the new records. The program then sends an e-mail to each field center describing the protocol problems with this batch of data. Each field center only receives e-mail pertaining to its staff. If a particular field center has no errors in a given batch, they receive no e-mail (no news is good news.) However, all of the mail sent is copied to the study coordinator, and the database administrator at our site.

Summary

The ability to link the SAS System with electronic mail opened a new series of possibilities for the distributed data entry systems we develop and support. It allows for automated feedback from users as they work on the system, allows for data-driven immediate communication with those remote users, and reduces the costs involved in supporting a remote application. The initial programming, while somewhat involved, yielded a reusable code module that can be easily exported to other, similar applications.

The ability of the original code module to handle routine cases was expanded to handle exceptions as well with a minimum of reprogramming. The module can be called from anywhere within the application.

The system independence of the SAS System E-mail interface allowed for easy adaptation for another application on a different platform. The ability to speed the communications between the coordinating center and the field centers has improved our ability to quickly diagnose problems, handle issues of user support, and facilitated the overall conduct of the project and the data entry associated with this project. The successful development and implementation of this method assures its continued use in new projects using distributed data entry.

Further inquiries are welcome to:

Derek Morgan
 Division of Biostatistics
 Washington University Medical School
 Box 8067, 660 S. Euclid Ave.
 St. Louis, MO 63110
 Phone: (314) 362-3685 FAX: (314) 362-2693
 E-mail: derek@wubios.wustl.edu

This and other SAS System examples and papers can be found on the World Wide Web at:

<http://www.biostat.wustl.edu/~derek/sasindex.html>

SAS, SAS/AF, and The SAS System are registered trademarks of SAS Institute, Inc. in the USA and other countries. ® indicates USA registration.

Any other brand and product names are registered trademarks or trademarks of their respective companies.