Paper 35-27

DISTRESS and PATCH: SCL to Support Remote Applications

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

Abstract

With the growth of the Internet and increased availability of high-speed broadband connections, it is easier to support remote applications built with the SAS® System. By using the electronic mail interface built into the SAS System, support can be provided to remote users much more rapidly. The two SCL modules to be covered in this paper demonstrate how we improved the effectiveness of the application maintenance team with regards to bugs, and provided a way to send routine application messages automatically from within the application.

Background

We develop and support data entry and management applications that have been built using SAS/AF® and SAS/FSP®, along with some customized shell programs. These applications are distributed to our field centers, where the data are actually entered. Each field center has a SAS System-enabled PC running under Windows, and all field centers have ethernet access. In previous studies, the data were transmitted via Federal Express, increasing the cost of the study. In addition, when problems were reported, the application maintenance staff had to use a slow, modem-based program to find the problem in order to diagnose and correct it.

Now that all of our collaborating field centers have high-speed internet access, we decided to improve several things about our software. First, electronic transmission of data was built into the system. We use the simple method of using the CALL SYSTEM statement in SCL to invoke an FTP program that employs secure socket transmission for security. Immediately, the cost of data transfer was reduced to almost nothing. However, our data management people would no longer get Federal Express envelopes that told them new data had arrived. How could a field center notify us that they had made a data transmission? The answer: by e-mail, of course.

Automatic E-Mail

Instead of forcing the person doing the transmission to notify us when they had sent data, why not have the data management application itself do the notification? Since we started SSH from within the application with a CALL SYSTEM, and with access to a MAPI32-compatible PC-based e-mail system, there was nothing to stop us from using the SAS System e-mail interface to let us know when the CALL SYSTEM had executed.

Using Electronic Mail via SAS Component Language

The SAS System electronic mail interface is deceptively simple. There are e-mail tokens that you use to dictate the actions of the mailing program. You use PUT statements to create what is essentially a text file that is sent to the mailing program. This text file contains the e-mail tokens you've used, and the text of your message, and it is sent to the mailing program for action. In SCL, FPUT and FWRITE are the commands that send text to an external file. The tokens still have to be used in the same manner, but otherwise, it's exactly the same process.

Inside The Application

The addressee list was hardcoded into the application, since the personnel responsible for this project had already been identified. We created a SAS System table containing machine-specific registration information. One of the columns is the name of the field center, which allows us to know who sent the data. We put this information in the subject line: "Data sent from *xxxxxxx*". Now, when the data are transmitted, we get an e-mail message from the field center, with the name of the file in the body of the message. That's a routine case, and is very useful in the compilation of the data, but it is not the only use of e-mail from within the application.

When something fails inside the application, we also receive electronic mail. There are certain key events within the application that can flag abnormalities. In this case, it is a summary file that should be updated each time data entry ends. However, an abnormal termination of data entry causes the update operation to fail. Since the update operation is centrally located in the code, the application watches for this failure. If the operation fails, the application sends an e-mail to the maintenance group. The e-mail consists of a title line stating the nature of the problem, giving the maintenance staff a general idea of where the problem is located. Then, the session log file is included as an attachment to the e-mail.

This allows the maintenance team a good look at the state of the application and a chance to see what process was running when the problem occurred. End users can tell you what they were doing at the time of the problem, but they cannot tell you what the program was doing. Between these two pieces of evidence, defect resolution time should decrease. Our experience with this is slowly confirming this; the diagnosis of problems is faster in this project, leading to quicker resolution.

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

The following SCL code details how all e-mail is sent from the application. By passing parameters to the SCL module, it handles sending the transmission notification, abnormal program execution notification, and a software patch notification. The last case will be discussed in detail later in this paper. Everything realted to sending the email message is here, including the command to attach the SAS System program log.

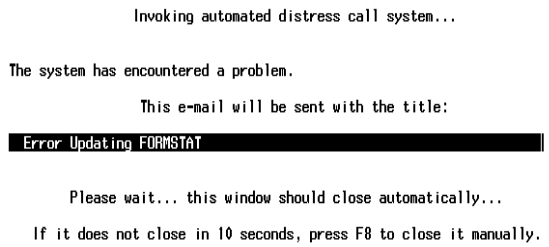
```
  /* DISTRESS.SCL ╥ Send e-mail to Coordinating
  Center From Data Entry System.  Get title and
      log file name from calling module */
1   ENTRY title $ 80 logfilename $ 80;
2   INIT:
           /*  Set up recipients  */
3   prog = "derek@wubios.wustl.edu";
4   copy = "notreal1@wubios.wustl.edu";
5   LENGTH site_id $ 10;
6   disptitl = title; /* Set title field in
    display window */

/* Set reason field in display window */
7   IF INDEX(title,"Snapshot sent") THEN DO;
8      errorkey = 0;
9      reason = "the snapshot has been sent.";
10  END;
11  ELSE DO;
12     errorkey = 1;
13     reason = "there is a problem.";
14  END;
15  REFRESH;  /* Refresh display window */
16  rc = rc;

      /* Get Field Center Information from
            registration table */
17  dsn = "hgx.register (READ=" ||
          SYMGET('rpass') || ")";
18  register = OPEN(dsn,'I');
19  CALL SET(register);
20  rc = FETCH(register);
21  rc = CLOSE(register);

22  MAIN:

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

   /* Open output file for e-mail creation */
24  mailfile = FOPEN('HELPME','A');
25  IF mailfile THEN DO;
26     rc = FPUT(mailfile,"!EM_NEWMSG!");
27     rc = FWRITE(mailfile);
28     rc = FPUT(mailfile,"!EM_TO!"||prog);
29     rc = FWRITE(mailfile);
30     rc = FPUT(mailfile,"!EM_SUBJECT!"||
                 title);
31     rc = FWRITE(mailfile);
32     IF SYMGET('userid') NE 'PROG'THEN DO;
33       rc = FPUT(mailfile,"!EM_CC!"||copy);
34       rc = FWRITE(mailfile);
35     END;
```

```
        /* Close log file in progress */
36     IF logfilename EQ ' ' THEN DO;
37        SUBMIT CONTINUE;
38          PROC PRINTTO;
39          RUN;
40        ENDSUBMIT;
41        logfilename = SYMGET('rootlib') ||
                        "\tmp\hgendes.log";
42     END;

     /* If it's an error, attach the log */
43     IF errorkey THEN DO;
44        CALL WAIT(2);
45        rc = FPUT(mailfile,
               "!EM_ATTACH!"||logfilename);
46        rc = FWRITE(mailfile);
47     END;
48     ELSE DO;  /* No problem, just data */
49        mailtext = site_id ||
          " has sent their snapshot.";
50        rc = FPUT(mailfile,mailtext);
51        rc = FWRITE(mailfile);
52     END;
53     rc = FCLOSE(mailfile);
54  END;
55  ELSE  /* display msg if mail not sent */
56     _MSG_ = SYSMSG();

           /*clear E-mail filename */
57  rc = FILENAME('HELPME',' ');
58  CALL WAIT(5);

59  TERM:
60  SUBMIT CONTINUE;  /* Resume logging */
61  PROC PRINTTO LOG=logfile;
62  RUN;
63  ENDSUBMIT;
64  CALL EXECCMD('CANCEL');  /*Close Window */
65  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. It 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 passed as parameters to the module (line 1). The log file name is left blank if the session log is to be attached. The recipients of the mail generated here are already known, so their addresses are hardcoded in lines 3 and 4. The e-mail address in line 4 will be put in the CC: field of the e-mail. Alternately, a recipient list could be generated from a data table. The display window (fig. 1) is set up in lines 7-16, and it contains 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 (line 17-21), so that the recipient knows who sent the mail.

Figure 1: Display Window



MAIN is where the e-mail is assembled. FPUT and FWRITE statements are used to produce the text in the external file, and to write the e-mail tokens there as well. The !EM_ATTACH! mail token is executed only if a log is to be attached to the mail. A simple line of text is written for routine data transmissions. Lines 32-35 prevent the "CC:" field from being used during development, debugging, or bench testing, so that the rest of the team doesn't get e-mail from the application. The session log is closed for attaching in lines 36-42 with PROC PRINTTO. You cannot take the an active session log from the log window, so it is necessary to change the log file destination when the application is started by using PROC PRINTTO in the autoexec.sas file.) The TERM section resumes the session log file and closes the display window.

The Other Problem

Sometimes it's necessary to execute a SAS program as a part of an update to an existing application. However, with the migration of our applications from OS/2 to Windows-based platforms, we ran into a minor problem. Where it had been easy to execute a SAS program through an OS/2 command file, the new path to sas.exe was just a little too long to execute in an MS-DOS batch file. While there are easy ways around this, the increased security of our datasets (involving SAS System encryption) ruled out those solutions. After all, what good does it do to have encryption passwords stored in an ASCII file?

This is why the PATCH module was developed. It checks for the presence of an update, then determines if there is a SAS program to be executed. If so, it will execute it, and then return the user to the application. Since macro variables are used to provide passwords within the application, we use macro variables to replace the actual passwords in the code. This module is called from the application's user authentication module.

PATCH uses a "version control dataset" (VCDS) to trigger its activation. This dataset has one variable and one observation, which is the current version number of the application. A similar dataset contains the previous generation of the VCDS. If they are the same, PATCH does nothing.

When they are different, PATCH checks to see if a file matching the current version is in the patch subdirectory. If not, it just updates the generational VCDS. This stops PATCH from executing again until the VCDS is changed. If this patch file does exist, then it is submitted to the application for execution using a SUBMIT block and a %INCLUDE statement. It uses DISTRESS to send email certifying the application of the upgrade, so that if a program is executed in this session, the SAS log for that patch program will be included. This keeps technical staff abreast of the status of the application at each remote location without having to ask end-users if they installed the update.

```
1   ENTRY version $ 16;
2   INIT:
  /* Get version number from VCDS, remove non-
              numeric characters */
3   version_no = COMPRESS(version,"()
    abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQR
    STUVWXYZ.");
      /* Check for a patch program file */
4   filestr = "\patch\patch" ||
    TRIM(LEFT(version_no)) || ".sas";
5   rc = FILENAME('patchfil',filestr);
6   chk = FILEREF('patchfil');
 /* No patch file - update generational VCDS */
7   IF chk LT 0 THEN DO;
8      SUBMIT CONTINUE;
9        OPTIONS SOURCE2;
10       DATA gcx.vcds_1 (WRITE=&wx ALTER=&ax);
11       SET gcx.vcds;
12       RUN;
13       OPTIONS NOSOURCE2;
14  ENDSUBMIT;
/* notify CC that update has been applied */
15     problem = "DES Upgrade to " ||
                 TRIM(LEFT(version)) ||
                 "installed";
16     CALL DISPLAY('gc.cat.distress.program',
                    problem,' ');
17     GOTO term;
18  END;
  /* patch file exists, include and run it  */
19  CALL WAIT(2); /* For display purposes */
20
21  SUBMIT CONTINUE;
22     OPTIONS SOURCE2;
23     %INCLUDE patchfil;
24     DATA gcx.vcds_1 (WRITE=&wx ALTER=&ax);
25    SET gcx.vcds (READ=&rx);
26    RUN;
27    OPTIONS NOSOURCE2;
28  ENDSUBMIT;
29  CALL WAIT(2);
    /* patch file ran, return SAS log to CC*/
30  problem = "Patch " || TRIM(LEFT(version_no))
    || " applied";
31  CALL DISPLAY('gc.cat.distress.program',
                  problem,' ');
32  GOTO TERM;

33  MAIN:
34  TERM:
35  RETURN(0);
```

How Does This Work?

The value from the version control dataset (VCDS) is passed to PATCH as a parameter. Since the value of the version may contain characters, (e.g., "0.9 beta") it is compressed to the numeric value in line 3.

The patch file, if it exists, will be located in the "patch" subdirectory, with a name of "patch**xxx**.sas", where **xxx** corresponds to the version number of the application. Lines 4-6 check to see if this file exists. If it does not exist, the current VCDS is copied to the generational VCDS, and email is sent to technical staff to let them know that the application upgrade was applied (lines 8-18).

If it does exist (lines 19-31), the patch file is sent for processing in a SUBMIT block. The title of the email message relayed through DISTRESS is slightly different, and the SAS log of the %INCLUDEd file is attached to the message. Finally, the current VCDS is copied to the generational VCDS, and the module closes.

Summary

DISTRESS can notify technical staff of problems with an application before it shows up in technical support calls and/or corrupted data. Its ability to send SAS logs in the e-mail message it sends is an invaluable aid in diagnosing problems that occur in a distributed application.

PATCH reports that any remotely distributed system patch was applied, and executes a SAS program if necessary. It uses DISTRESS to report the system upgrade back to technical staff, along with any SAS log that is generated.

PATCH and DISTRESS may not be incredible advances in SAS technology. Their utility overrides their simplicity, allowing technical staff to stay in close contact with the end-users without the intervention of those end-users.

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