

Use SAS/AF, SCL and MACRO to Build User-friendly Applications on UNIX

Minghui Yang, Ph.D, Boeing Logistics Market Research

0. Introduction

In the business application environment, many business analysts know their business very well. But they may not have the right tools in their analysis. Although SAS is a great tool for business analysis, some business analysts do not have enough experience in SAS programming. They have some knowledge about SAS and SAS applications. And they know WINDOW applications very well. But on UNIX, they may not have the user-friendly tools that SAS PC has. How to use SAS to build user-friendly applications on UNIX is a challenge job for a SAS developer. The author participated in a big project for MCI to use SAS/AF, SCL, and MACRO to build a user-friendly application for investigators to do business analysis on UNIX environment. The project was successful and the author would like to share the experience with other SAS users.

1. Project Description

In the old environment, the application was on mainframe and written by COBOL. All interfaces were written by CLIST. So the investigators can use those interfaces to look at the calling records, analyze the data, and correct the problems. In the recent years, MCI has planed to use UNIX to store the calling records and do data analysis to those calling records. The calling records are stored in the binary format. Each calling record is called an OSCIR which may contain up to 27 types of segments with variable lengths. It means one OSCIR may have several hundreds of segments; all segments in each OSCIR do not appear by the numerical order; and the same type of segments may have different lengths. For example, we may have segment 10 in front of segment 1. We may have several segment 9: the first one may have length 64, and the second may have length 87. If all records are binary data and one file contains millions of records, we have a huge binary data to read. Reading binary data is not very common in the business application environment compared with reading other format of data. However, we have to read binary data in this project. There are several difficulties:

- The huge data in one file was only one-line long. So we have to read every single byte correctly. If the data was not read correctly even in one field, the rest of the data read by the program may become garbage. It is much easier for reading other text files by using some of the SAS *infile* statement options '*delimiter=*', '*mssover*', and '*dsd*'. If one line was not read correctly (say the data entry was wrong), the rest of data may still turn out pretty good.
- To debug the program is much harder. We need to have a test data with all segments which have all types of variable lengths to test the SAS program. Unfortunately, such a test data did not exist. So the author must design the program to be very flexible to read all types of segments.

- The third challenge is the speed. SAS program needs to read the data, separate each segment, and separate each OSCIR at the right position. If the algorithm is not efficient, the speed is slow.
- The fourth challenge is to build user-friendly interface for investigators who are familiar with the data, but not familiar with SAS programming. If this part is not done correctly, the project will fail because the users are investigators but not SAS programmers. If they do not like the tool, they will not use it. So it is very important to build a user-friendly application for them. The other parts of the project are behind the seen. Only this part can be highly seen by all people.

2. Read Binary Data on UNIX by SAS

When the author started the project, the first task was how to read the data correctly. The author used SAS to read ASCII files and other text files for a long time. SAS is really good for reading those files. But when the author used SAS to read binary files with variable length, the problem occurs because there were very little reference materials available.

Not many SAS programmers know how to read one-line long binary data, especially with variable length. The only reference material available was a test SAS program developed by some SAS consultants. In the test SAS program, the following methods were used:

- Read a 200-byte character string each time (SAS's current limitation for a character string). If the file is 400,000 bytes long, 2,000 strings are read;
- Use SAS function SUBSTR to read each field in each string;
- If the field is a numerical variable, use SAS informat S370FIBw.d or S370FPIBw.d to convert the binary value to a SAS value.

This program is not efficient. First, SAS has to read a character string, then use string function, then convert it to a SAS variable. It is slow. The second and the most serious problem is this program can not deal with the variable length records very well. Since at the position 200 of each string, any segment and any variable in the segment can occur. The program has to use so many "if" condition statements to figure out which segment was read and which variable it was. This is not the end of our story. The position 200 may split one variable to two parts. The part one is in one string and the part two is in another string. To put those two parts back into one piece and convert it to either a numerical value or a character value is a nightmare. It is not surprised that the test program, with more than 16 pages long (1,000+ lines), only reads 5 simple segments. And it is slow. A change must be made.

The second problem is the SAS program has to read data on different UNIX machines. For example Digital UNIX machine is different from other UNIX boxes. The most significant bytes and the least significant bytes for a numerical value on a Digital UNIX machine is the opposite from other UNIX machines. The hardware difference may cause big problem for the SAS program if a different machine is used. For example, generally

you can use following SAS codes to read a 4 bytes numerical variable (binary format) and a 3 bytes character variable:

```
input X S370FIB4,  
      Y $3.;
```

But if you use Digital UNIX machine, the codes may become:

```
input Z $4.  
      Y $3.;  
X=input(reverse(Z), S370FIB4.);
```

The above method is not very efficient since the codes have to read a character variable first, then reverse it, then convert the binary value to a SAS variable.

Although a C/C++ program could be used to read the data, the program would be complicated. For every numerical value on a Digital UNIX machine, the C/C++ program needs to convert the bytes order. Many structures, unions or classes will be defined. If a segment changes or a new segment is added, to maintain this program may not be easy. Since SAS is the preferred tool for this project, the author needs to find a good way in SAS to read the binary data. In one and half months, after many failures and tests, the entire data was read correctly. The whole codes were completely re-written. There are lots of details which will not be given here. Only the design highlights are given below. There are four factors led the author to read binary data successfully:

- The first factor is to define 'infile' statement correctly. The infile statement is: infile filename lrecl=xxxxxx col=cc; where *filename* is user defined, *xxxxxx* is the file length in bytes, and *cc* is the column control variable (defined by SAS). You may add RECFM=FB in the option. This statement looks simple but not easy to get. The SAS variable *cc* is like a cursor control to read the file. The *lrecl* must be set as the file length to read the data correctly. Please remember infile statement is the most important foundation to read a binary file.
- The second factor is to select the correct 'informat' statement. The SAS codes (informat for input data): input X IBw.d was used to deal with Digital UNIX machine. No other informat or functions are needed.
- The third factor is to write the 'while-loop' and 'do-loop' thoughtfully. Two while-loops were used. The position in the input file is controlled by the column variable *cc*. The program stops when the file length is reached by *cc* in the first while-loop. The second loop ends when an OSCIR reaches its length. If the file does not reach the end, another loop to read the next OSCIR starts. You can think about *cc* is like a cursor moving from the left to the right on a long record. Since each OSCIR and each segment have a header respectively, when *cc* reads the OSCIR header to obtain the OSCIR length, it "knows" how far the cursor can be moved for this OSCIR. Within this OSCIR, *cc* "reads" segment header first and "knows" how far it can move for this segment. If this OSCIR length does not reach, *cc* will move to read next

segment. Output will be made after each segment ends. There are two counters: a counter for OSCIRs and a counter for segments in each OSCIR. The segment counter needs to be reset after each OSCIR reaches end.

- The fourth factor is to debug and check the codes and output thoughtfully. Eight MACROS programs were developed to reduce the repeat codes. The variable length for each segment is dealt within each segment by MACRO functions. For example, if segment 9 has two lengths, the program uses MACRO programs to control the reading (which format will be used). MACRO programs simplify the reading process and increase the efficiency.

The entire codes, reviewed by SAS Institute, are very efficient.

- The total length is about 900 lines, 1/6 of the original one. We do not need to read 200 bytes each time and to do many conditional checking for the nightmare;
- The speed is fast. It reads 200,000 OSCIR records (each OSCIR is about 1,400 bytes) in less than 9 minutes.
- Since many MACROS are used, it is easy to maintain the codes for adding or deleting segments in the program.

3. Use SAS/AF, SCL and MACRO to Build Interface on UNIX

During the last five years, Window applications become very popular. If an application can stay in the software develop business, it is likely user-friendly and Window-like. The SAS View on the PC is user-friendly. SAS/AF is another effort made by SAS Institute to make SAS more user-friendly on PC, UNIX and mainframe. As well known, SAS/AF is not a new SAS product and SAS/ASSIST is an application of SAS/AF. If you know Visual Basic or Visual C++ well, you will notice immediately that there are some similarities between VB and SAS/AF, SCL. But SAS/AF and SCL are not as easy to use as VB is. Also, the interface SAS built is not as fancy as VB does. The author does have certain reservations for SAS/AF applications on UNIX. SAS still has lots of work to do in this area. A good WINDOW programmer may not like SAS/AF first, but may feel SAS/AF is OK after building some applications.

This article is not a SAS/AF and SCL tutoring material. In the project author participated, there were lots of SAS/AF frames built by several SAS developers. The author can not give much details about the entire developing work and would like to show just one example about the SAS/AF application. Don't be frustrated about the SAS/AF applications. If you know MS VB, it probably much easier for you to build a pretty good SAS/AF frame. If you know the base SAS well, it is not hard for you to learn SCL codes.

The application is to use SAS/AF frame to build an interface which can send e-mail to SAS users. The investigators can use this tool to report data problems or to ask questions about SAS. After logon, the user names were checked and kept by UNIX. The date is generated by the system. If user's information is stored in a system file, the user's phone number, e-mail address and department location can appear in the SAS/AF frame

automatically. However, in the current version, the end-user has to enter his/her phone number, e-mail address and department location. The frame looks as follows:

SAS: AF
Send us a Message

[Goback](#) [Help](#)

Your Name: Date:

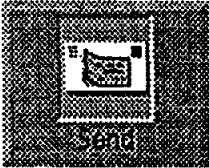
Your Phone Number (VNET: 622-5668):

Your E-Mail Address:

Your Dept/Location:

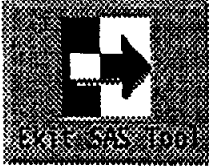
priority:
 ◆ 1 (Urgent, - One Business Day)
 ◇ 2 (Normal, - Two Business Days)
 ◇ 3 (Low Priority, - Five Business Days)

Problem Descriptions (limited to 3 lines):



Send

We can also be paged at:
 1-800-PAGE-MCI
 PIN# 168 44 10



Call SAS 100

The SCL codes are:

```
init:
  dates=today();
  submit continue;
  filename name pipe 'whoami';
  data user;
    infile name;
    input username$;
    username=trim(username);
    usrl=length(username);
    call symput('username',username);
    call symput('usrl',usrl);
    put username;
```

```

run;
data _null_;
    usrl=symget(usrname);
    username= symget(usrname);
    length usrn $&usrl;
    usrn=username;
    call symput('usrn',usrn);

run;
endsubmit;
return;

subm:
    submit continue;
    filename outbox EMAIL "&email";
    proc format;
    value prio
        1='Urgent, - Within One Business Day'
        2='Normal, - Within Two Business Days'
        3='Low Priority, -Wwithin One Business Week'
    ;

run;
data _null_;
    length name email dept phone $40 problem $200;
    format dates date9. Priority prio.;
    dates=&dataes;
    name="&name";
    usrn="&usrn";
    email="&email";
    dept="&dept";
    priority="&priority";
    phone="&phone";
    problem="&problem";

    file outbox
        to="USERGROIUP//list@projectteam.com"
        cc="&usrn"
        subject="** Production Problems **";

    put name=;
    put dates=;
    put email=;
    put dept=;
    put priority=;
    put phone=;
    put problem=;
run;
endsubmit;
call display('tools.apps.sentmsg.framer');
return;

endsas:
    call execcmd('endsas');
return;

term:
return;

```

After looking at above codes and graph, you probably notice:

1. SCL codes are similar to the base SAS codes. However, SCL has many its own statements and functions such as 'submit continue', 'endsubmit', 'execcmd', 'call display', 'varnum', 'vartype' etc.
2. A SCL program should have 'init', 'main', and 'term' three sections and other sections which related to the 'objects' in the frame.
3. To pass certain variable values, macro variables need to be defined. So some functions at SAS MACRO such as 'symget', 'symput' will be used.
4. On the UNIX environment, some codes are for UNIX only such as 'pipe', 'whoami' etc.
5. PROC BUILD should be used to build the frame

Since many SAS users know Windows application well, we like to mention about some differences between SAS/AF and MS VB, SCL and base SAS.

1. MS VB can build frame fast. Without writing codes, the frame looks like the real interface. It can have radio buttons, list boxes, and other icons. Because they are the objects. And all Windows applications are object oriented. Similarly, a SAS/AF frame contains 'regions' and 'objects' including icons, text boxes etc. But compared with VB frame, a SAS/AF frame may look very simple. It may only contain icons. Although SAS already built several hundreds of icons, you may not like the style, the color, the fonts. In fact, SAS/AF does not have many choices in icon shape, color, and fonts. So the frame built by one SAS user may look very similar to the one built by another SAS user. The size of a SAS icon is quite big and can not be reduced to a very small one to fit your needs (at least so far on the UNIX). So when you build a SAS/SF frame, you need to think about how many sub-frames or reports you are going to have. Because of the size restriction, you can not arrange many objects in a frame. If you do have many reports or many tasks in an application, you may need to build several frames including 'parent' frame and 'child' frame.
2. Each VB frame has VB codes associated with the frame. Similarly, a SAS/AF frame generally has SCL codes associated with it even though the SCL codes are not required for each frame. SCL codes look like other SAS codes, but could contain some functions which base SAS does not have. For example the function 'exist' can help the developer to determine if a file or dataset exists before rest of programs is called. Each icon on the frame may correspond a block of SCL codes. The name of the block is the icon name you have defined. This is quite similar to VB. The author recommends using '%include' statement in some of the SCL codes for some SAS reports because testing and development is much easier. In the SAS/AF frame development, the debug and compile is not as easy as in the base SAS. If the SCL codes use some input data to generate reports, it is better to develop the codes first in the base SAS. Then the codes can be included in the SCL. Please include the 'libname' for the program directories in the 'autoexec.sas' file. So the SCL program 'knows' where to find the SAS program.
3. MACRO programs and other MACRO variables can be used in the SCL program.

4. When select variables from 'list box', the number of variables is limited to five which is not very convenient in some cases. Also the variable type (character or numerical) determines which SCL function should be used. This may cause inconvenience.
5. To test a frame 'TESTAF' command is used. But this command can not be used if the FRAME entry is in a library accessed with SAS/SHARE software or the SCL codes have SUBMIT statements. You should issue SAVE command and use AF command to test. This may cause inconvenience.

4. Conclusions and SAS Limitations

SAS still has limitation for writing SAS data back to a binary file. SAS can read binary data in the bit level, but can not write it back in the bit level (SAS can read and write binary data in the byte level). Sometimes, this may not be very convenient for the users. For example, if a user reads the binary data in the bit level, then modify the data, the modified data is extremely difficult to be written back to the binary format. Unless the data has very limited range, say only has three values, you may use conditional statements to write the data back to a binary format. In such an application, C/C++ program probably has to be used.

Also SAS is not very fast to read binary data since the informat *IBw.d* may call several C functions to read the binary data.

SAS/AF frame might be simple and not very fancy. The reference menu is not well written.

However, SAS is a great tool to read binary data. After reading, the data is in SAS format. It is great for the further data analysis and statistical analysis.

SAS/AF is also a good tool to build some simple or not very complicated frames to make the SAS application user-friendly. But the development cycle might be long. Also SAS/AF programmers are not easy to be found. The project leader needs to think about maintenance issues for the development.