_UPGRADING USER-WRITTEN PROCEDURES FOR VERSION 5 SAS® SOFTWARE:
A USER'S PERSPECTIVE

Richard L. Gimarc, Boole & Babbage, Inc.

1. ABSTRACT

The services provided by the SAS System for user-written procedures were significantly enhanced in version 5. Although pre-version 5 procedures still execute under version 5, the new version provides the procedure writer many new services. This paper describes the author's experience upgrading a user-written procedure. Special emphasis will be placed on upgrading custom procedure statements to the version 5 style.

2. INTRODUCTION

Extensibility is a term that characterizes the SAS System. What this means is that the SAS System provides documented interfaces which can be utilized to expand the capabilities and functionality of the SAS System. One of the methods available for extending the SAS System is user-written procedures. These procedures are invoked with the familiar PROC statement. Other procedure statements may be used depending on the design of the procedure.

There are several reasons why you might choose to package your code in a user-written procedure rather than using a SAS macro or %INCLUDE code:

1. To provide an interface to a non-SAS data base system.
2. The calculations use proprietary algorithms.
3. The functionality being added is beyond the scope of the SAS language.

Procedure writing is not an esoteric subject restricted to a small handfull of SAS System users. There has been a steady stream of papers presented at SUGI conferences describing user-written procedures. A scan of the past five SUGI proceedings produced the following paper counts:

SUGI 8 (1983): 11 papers
SUGI 9 (1984): 12 papers
SUGI 10 (1985): 10 papers
SUGI 11 (1986): 11 papers
SUGI 12 (1987): 7 papers

Furthermore, there is a SAS System document [SAS86b] which describes user-written SAS procedures contributed to, and distributed by, SAS Institute.

Due to the change in the style of user-written procedures in version 5, there have also been several papers presented at SUGI conferences by SAS Institute personnel describing how to write a procedure in the version 5 style [Betaa86] [Park86] [Hin87a] [Hin87b].

These papers, along with [SAS86a], provide an excellent set of reference material on version 5 user-written procedures. However, there are some notable deficiencies:

1. All the papers have been presented by SAS Institute personnel. There has not been a paper presented from a user's perspective.
2. There is no record of experience regarding upgrading pre-version 5 procedures to the version 5 style.
3. The implementation of non-list type (custom) procedure statements has not been fully described.

This paper will attempt to fill these deficiencies by describing the author's experience upgrading a user-written procedure to the version 5 style.

3. ENVIRONMENT AND TERMINOLOGY

This paper describes procedure writing under the MVS operating system using the PL/I programming language.

The term old SAS will be used to refer to the SAS System prior to version 5. Likewise, new SAS will be used to refer to version 5.

4. PROCEDURE WRITING

A user-written procedure consists of two modules: the parser module and the procedure module. The parser module is responsible for examining the SAS statements entered by the user. These statements are checked for valid syntax and the information entered by the user is saved in internal statement
structures for later use by the procedure module. Once the parser completes its task the procedure module is given control. The procedure module will use the information saved in statement structures by the parser to determine what data sets, options, and parameters were specified by the user. Using this experience (SAS81), especially if you chose to write a procedure with non-list type procedure statements. List type statements such as BY and VAR were easy to implement, parser macros were available for defining the format and syntax of your list. Non-list statements, however, required a great deal more work. The procedure writer was responsible for examining the syntax of the statements himself with his own code [Gim84]. And depending on the syntax of the statements defined for the procedure, this could be a rather complex task.

5. AN EXAMPLE: UPGRADE PROC TOKEN

The user-written procedure PROC TOKEN was presented at SUGI 10 [Gim85]. This procedure enables you to evaluate the performance of a token ring local area network. We will examine how portions of this procedure were implemented using old SAS, and then look at how the implementation was greatly simplified using new SAS.

The syntax for PROC TOKEN is given below:

PROC TOKEN OUTPUT=SASdataset  
  TOLERANCE=number  
  MAXIT=number;

RING NAME='ringname'  
  STATIONS=number  
  OVERHEAD=number tunit  
  CAPACITY=number tunit  
  MSGSIZE=message_size_specification  
  ARRIVAL=arrival_rate_specification;

5.1 THE PARSER MODULE

All SAS procedures have a pre-defined syntax which describes the format of the SAS statements used to invoke the procedure.

We will focus our attention on the implementation of the parser for the OVERHEAD and MSGSIZE parameters of the RING statement.

The OVERHEAD Parameter

A time interval is specified with the OVERHEAD parameter. The procedure module was designed to perform its calculation based on a time interval specified in seconds. However, to make the specification of this interval more flexible, the procedure allows the specification of a time unit, tunit. Valid time units are SEC for second, MSEC for millisecond, and so on.

Old SAS required the procedure writer to create his own parser for parameters such as OVERHEAD. The parser had to examine the sequence of characters entered by the user and verify that they conformed with the syntax defined for the statement. Concurrent with syntax analysis, the parser had to save the values entered by the user so that the procedure module could use them later.

Compiler theory advocates the use of a grammar to describe syntax [Aho79]. A grammar is simply a formal notation used to give a precise specification for a language (very similar to a mathematical formula).

A grammar was developed for the OVERHEAD parameter and is shown pictorially in Figure 1 in the form of a transition diagram. This grammar indicates that the character string "OVERHEAD" must be followed by an equal sign, a number, and an optional time unit. Code was written to ensure that the information entered by the procedure user conformed to this syntax.

To understand how this grammar is used, we will manually parse the following OVERHEAD specification:

OVERHEAD=35 MSEC

The parser begins in state S1 where it expects to see the character string "OVERHEAD" followed by an equal sign. The grammar then indicates that the next item entered by the user must be a number. Once we reach state S4 we have a choice of what comes next. The grammar states that the next item in the input stream must be one of the valid time units.

In addition to verifying syntax, the parser needs to save the values entered by the user so that the procedure module can use them to perform its computation. For example, when the parser reaches state S4, it knows that the number it just scanned is the unitless value for OVERHEAD. This number (35) will be saved for later use. Also, as the parser scans the time unit and transits from state S4 to S5, it knows what time unit was entered by the user. In this example, a flag will be set to indicate that the user specified MSEC.

New SAS introduced a powerful facility for symbolically describing a grammar and letting the SAS system examine the syntactic structure and
save user entered values itself. The procedure writer no longer has to examine the sequence of characters entered by the procedure user to verify statement syntax.

The portion of the new SAS grammar describing the syntax for the OVERHEAD parameter is shown in Figure 2.

OVERHEADP is the name of the grammar production for parsing the OVERHEAD parameter. As before, the parameter begins with the character string "OVERHEAD" and is followed by an equal sign. Next, a number is required followed by an optional time unit. @PARM is a semantic action which will save the specified NUMBER in a statement structure so that the procedure module can access it later. The time units listed between the brackets < and > are optional. If one is present, an option flag is turned on to indicate which one was specified.

Notice the similarity between the old SAS and new SAS grammars. They both accept the same syntax for the OVERHEAD parameter. But with new SAS, the burden of syntax analysis has been shifted away from the procedure writer. The procedure writer no longer has to inspect the input himself, he simply lets the SAS parser do the work. Once parsing is complete, the procedure module then accesses the statement structures built by the parser to determine what was specified by the user.

The MSGSIZE Parameter

A more complex example is the MSGSIZE parameter. This parameter is used by PROC TOKEN to specify the average size of messages to be transmitted on the token ring in terms of its mean and standard deviation. The following three forms of the MSGSIZE parameter are valid:

(a) MSGSIZE=(mean, std dev)
(b) MSGSIZE=mean
(c) MSGSIZE=mean

In (a) the mean and standard deviation are specified explicitly. If the standard deviation is zero, form (b) may be used. And if the standard deviation is equal to the mean, form (c) may be used.

The grammar developed to parse the MSGSIZE parameter in old SAS is shown pictorially in Figure 3.

This grammar contains three transition diagrams which are nested. After scanning the character string "MSGSIZE" and the equal sign you reach state S3. From there, you call the transition diagram labeled MSGSIZEP1 and continue the scan. Nested transition diagrams work just like subroutine calls. When you reach the final state in a called transition diagram, you simply return to the calling diagram. So after MSGSIZEP1 completes, the parser would return to state S4 in transition diagram MSGSIZEP.

Given the sequence of characters entered by the user, the user—written parser was able to determine whether the syntax was valid. And in addition, as the syntax was checked, it was easy to determine which form of MSGSIZE was used by noting which arcs were traversed in the grammar's transition diagrams.

Again, new SAS has made the parsing of a parameter like MSGSIZE easy. The new SAS grammar is shown in Figure 4.

The grammar is more complicated than that for OVERHEAD, but the idea is the same.

The production MSGSIZEP says that following "MSGSIZE=" you must use production MSGSIZEP1 (this is like calling a transition diagram in the old SAS style). MSGSIZEP1 says that following the equal sign you must either have a number or a left parenthesis. If it is a number, then the number is saved in a statement structure using the OLIST semantic option and option flag 9 is turned on to indicate that this is a form (c) specification. If the next item scanned was a left parenthesis instead of a number, then the MSGSIZEP2 production must be used to continue evaluation (form (b) or (c) was used).

MSGSIZEP2 says that following the "(" it expects to see a number which it saves using OLIST. Next, if it sees a ")", then option flag 10 is turned on to signal form (b). Otherwise, there must be a comma, followed by a second number, and then a right parenthesis.

During the parsing, the values entered by the user are saved in a list anchored in the statement structure using OLIST. The list may either have one or two numeric elements depending on what form of MSGSIZE was used. And option flags 9, 10, and 11 indicate which form of MSGSIZE was used.

The grammar defined in new SAS exactly matches the user—written grammar used in old SAS. As in the previous example with OVERHEAD, the new SAS grammar allows the procedure writer to ignore the details of syntax analysis and concentrate more on the functionality of the procedure.

516
4.2 INITIALIZING AN OUTPUT DATA SET

One of the results of executing PROC TOKEN is a SAS output data set. This data set is identified by the OUTPUT parameter on the PROC statement and contains performance metrics describing the token ring model you executed.

The output data set is initialized by the procedure module using service routines provided by the SAS System. The sequence of service routine calls required to prepare the output data set in old SAS is shown in Figure 5.

You begin by calling LODOUT to load the routines necessary for writing to a SAS data set. Next, SETDSN is called to designate the OUTPUT= data set as the current data set. All subsequent service routine calls will operate on the designated current data set. The output data set is then opened using OPOUT. For each variable to be included in the output data set, ONAMES is called. Parameters are passed to ONAMES giving the variable name, format, label, etc. And finally, ENDNAM is called to terminate variable definition and ABUF to allocate a data set buffer.

The new SAS code required to initialize an output data set has been simplified. This code is shown in Figure 6. Initialization begins by calling XVPUTI which initializes the process that writes observations to a SAS data set. Next, XVPUTD is called to define the output data set variables. The call to XVPUTE terminates variable definition and output data set initialization.

Note the similarity between the old SAS and new SAS code. They both perform the same function and the sequence of operations is the same. However, the number of different service routines which must be called has been reduced from six to three. Also note the service routine naming standard adopted in new SAS. The new SAS service routine names all begin with the prefix XV. This naming standard implies that the XV routines perform related functions (which, in fact, they do!). This contrasts sharply with the non-standard service routine names used in old SAS.

5.3 WRITING TO THE SAS LOG

The simple task of writing messages on the SAS log has also been simplified, especially if you are writing a procedure in PL/1.

Figure 7 shows the code necessary to write the following message on the SAS log (assuming the current date is March 10, 1988):

```
NOTE: TODAY IS 0310
```

In old SAS you were required to build the entire message yourself. That is, you had to construct a character string which contained the full text of the message to be written to the log. You also had to declare the length of the message separately. And if you were coding your procedure in PL/1, you had to declare and use a numeric data item to pass the message text with a numeric parameter list.

The new SAS implementation is again simplified. The new SAS XLOG routines shift the burden of message generation away from the procedure writer by providing dynamic message generation. The new SAS code is shown in Figure 7. The pound sign (#) is used in the text of the message to indicate where substitution is to occur. Also, it is no longer necessary to explicitly declare the length of the message in a separate variable (like LEN_NOTE01) or overlay define the text of the message with a numeric parameter list.

6. SUMMARY

Version 5 of the SAS System provides powerful tools for implementing user-written procedures. These tools enable the procedure writer to focus his attention on the functionality of the procedure itself, rather than trying to find ways to compensate for deficiencies in the procedure writing services.

The grammar processor of version 5 is an especially valuable, and welcome, tool. It drastically reduces the amount of time required to build the parser module for a procedure and provides powerful facilities for defining non-list type procedure statements.

The service routines available in version 5 eliminate many of the tedious tasks required of the pre-version 5 procedure. It will not be uncommon for procedures upgraded to the version 5 style to be much smaller. As an example, the number of lines required for the pre-version 5 implementation of parser/procedure module of PROC TOKEN was 662/1981. Under version 5, this was reduced to 119/936, a reduction of almost 50%.

And even though there is much talk about version 6 of the SAS System, all user-written procedures written in the pre-version 5 style will have to pass through the version 5 style if they are to be upgraded to the unannounced version 6 style.
7. REFERENCES

[Aho79] Aho, A. V., and Ullman, J. D.
Principles of Compiler Design.
Addison-Wesley, 1979

[Beta86] Betancourt, T. L.
Writing SAS Procedures. Proceedings of
the Tenth Annual SAS Users Group
International Conference, 1985

[Gim84] Gimarc, R. L.
Procedure Writing: How to Implement
Custom Procedure Statements.
Proceedings of the Ninth Annual SAS
Users Group International Conference, 1984

A SAS Procedure for Evaluating the
Performance of Token Ring Local Area
Networks. Proceedings of the Tenth
Annual SAS Users Group International
Conference, 1985

[Ihnen87a] Ihnen, L.
User-Written Procedures in Version 5.
Proceedings of the Twelfth Annual SAS
Users Group International Conference, 1987

[Ihnen87b] Ihnen, L.
User-Written Procedures in C.
Proceedings of the Twelfth Annual SAS
Users Group International Conference, 1987

[Park86] Parker, R.
How to Write SAS Procedures for
Version 5. Proceedings of the Eleventh
Annual SAS Users Group International
Conference, 1986

[SAS81] SAS Institute Inc.

[SAS86a] SAS Institute Inc.
SAS Programmer's Guide for PL/1,

[SAS86b] SAS Institute Inc.
SUGI Supplemental Library User's Guide,

518

8. AUTHOR CONTACT

The author may be contacted at:

Boole & Babbage, Inc.
1601 Rio Grande, Suite 335
Austin, Texas 78701
(512) 478-0788

SAS® is a registered trademark of SAS Institute
Inc., Cary, NC, USA.
OVERHEADP = "OVERHEAD" "=" number
< "SEC" | "MSEC" | "USEC" | "NSEC"

Figure 1: Old SAS transition diagram for OVERHEAD parameter.

OVERHEADP = "OVERHEAD" "=" OPARM(4,1)
NUMBER
< "SEC"
| "MSEC"
| "USEC"
| "NSEC"

Figure 2: New SAS grammar statements for OVERHEAD parameter.

MSGSIZEP: "MSGSIZE" "=" MSGSIZEP1

MSGSIZEP1: 

Figure 3: Old SAS transition diagrams for MSGSIZE parameter.
\textbf{MSGSIZEP} = "MSGSIZE" \
\textbf{MSGSIZE1} = ( "(" MSGSIZEP2 ) \\n\textbf{MSGSIZE2} = ( ")" \\n\textbf{MSGSIZE3} = )

\textbf{Figure 4: New SAS grammar statements for MSGSIZE parameter.}

\textbf{LODOUT} load routines to write data set  
\textbf{SETDSN} designate current data set  
\textbf{OPNOUT} open data set for output  
\textbf{ENDNAM} define variable in output data set  
\textbf{ABUF} terminate variable definition  
\textbf{Figure 5: Prepare output data set in old SAS style.}

\textbf{XVPUTI} initialize output process  
\textbf{XVPUTD} define variable in output data set  
\textbf{XVPUTE} terminate variable definition  
\textbf{Figure 6: Prepare output data set in new SAS style.}
DCL DATE BUILTIN;
DCL NOTE01 CHAR(22) INIT('NOTE: TODAY IS yymmdd'), LEN_NOTE01 FIXED BIN(31) INIT(22), NOTE01X FIXED BIN(31) UNALIGNED DEFINED NOTE01;

SUBSTR(NOTE01,17,6) = DATE;
CALL SASLOG(NOTE01X,LEN_NOTE01);

Figure 7: Write SAS log message in old SAS.

DCL DATE BUILTIN;
DCL NOTE01 CHAR(16) INIT('NOTE: TODAY IS #'),

CALL XLOG2(NOTE01,DATE);

Figure 8: Write SAS log message in new SAS.