WRITING SAS® PROCEDURES

Terri L. Betancourt, SAS Institute Inc.

ABSTRACT

Writing a SAS procedure requires a certain level of programming skill. The procedure writer needs to have a clear understanding of data structures, pointer manipulation, and dynamic storage allocation, as well as a working knowledge of the SAS System. Although oriented toward SAS users in the minicomputer environment, most of the ideas presented here will apply to other users as well. The intent of this paper is not to explain the technical information and detail required to write a SAS procedure -- that information is available in the SAS Programmer’s Guide. Instead, the attempt is to construct a conceptual framework upon which the programmer can build.

INTRODUCTION

Until recently, writing a SAS procedure on the IBM mainframes was difficult and often required knowledge of assembly language programming. User-written procedures on the minicomputers were not possible at all. Today, however, procedure writing is different. All SAS procedures can be written in a high-level programming language. Moreover, with the development of dynamic loading on the VAX, the Institute is now able to support user-written procedures in the VMS environment. Dynamic loading is also anticipated for the Data General AOS/VS and Prime PRIMOS environments.

There are several advantages to writing a SAS procedure rather than developing a stand-alone program:

- The SAS System provides many services which you would otherwise have to code yourself, e.g. syntax checking, data set management, and memory allocation.
- Using your procedure will be easy for other SAS users since they are already familiar with data input and data manipulation under the SAS System, as well as general SAS syntax.
- The data used by your procedure can easily be sorted, printed, and analyzed using other SAS procedures.
- Special features like BY group processing can be used.

A SAS procedure is simply a program (or group of programs), written in a high-level language that interacts with the SAS System to perform specified data manipulations. The basic functions of a SAS procedure are reading SAS data sets, performing data analysis, printing results, and creating other SAS data sets. The information and process control required for these basic functions is available within the SAS Supervisor.

The Supervisor is the part of the SAS System which maintains overall control of system processing. Therefore, your procedure will need to interact with the SAS System Supervisor. For that purpose, there is a set of routines, called X-routines, which act as an interface into the Supervisor. For example, to read a SAS data set your procedure might call XOPEN. To print a page of output you might call XPAGE. The knowledge of where and when to call a particular X-routine will become more clear as you begin to understand the internal design of SAS procedures.

Before your SAS procedure can interface with the SAS System, the following tools are needed:

- familiarity with the SAS System and its procedures
- a version of the SAS System which supports dynamic loading
- a working knowledge of a high-level programming language
- a thorough understanding of "overlay" techniques.

Since SAS procedures within the Institute as well as the SAS Supervisor are written in PL/I, all programs that interface with the SAS System must currently be written in PL/I. The Institute would like to eventually support user-written procedures written in other high-level programming languages.

Overlaying is a specialized use of the PL/I BASED attribute for data structures. However, a parallel concept exists in Pascal through the use of the CASE variant declaration and in the C-language through the UNION definition. The technique works as follows: a single memory location is used to store multiple types of information. That is, sometimes character data and other times numeric data, are stored in the same location. Overlaying is the means by which the appropriate attribute is assigned to the memory location in order to access the data in the correct form.

Once the tools are available, the following steps are taken in writing every SAS procedure:
1. Design the function and form of the procedure.
2. Write a grammar to specify the syntax of the procedure.
3. Process the grammar with the Parser Generator.
4. Write and compile support routines for the procedure.
5. Link the procedure with the SAS System.

Step 1: DESIGN

In order to design your procedure, you will need to answer many questions.

- What do you want your procedure to do for the user?
- What will be the name of your procedure?
- What options will be available on the PROC statement?
  - Which options will be on/off type options, and which options will have values assigned to them? Options which have values assigned to them are called "parameters." For each parameter, you will need to decide what types of values will be acceptable.
  - Will your procedure require an input SAS data set?
  - Will your procedure create an output SAS data set?
  - Which standard SAS procedure statements will be available within your procedure, e.g. VAR, ID, BY? And what types of variables will be allowed in these statements?
  - Will your procedure need procedure statements other than the standard statements?

In general, before you can get started, you must decide exactly what you want your procedure to look like to the user, which features are optional and which features are required, and what your procedure will do with the data that it is given.

Step 2: WRITING A GRAMMAR

Once you have a functional design for your procedure, the next step is to specify the procedure's syntax in a more formal way. Every SAS procedure follows a general form. The first statement of a procedure, the PROC statement, has the following possible features:

**PROC PRINT DATA=TEST SPLIT=* DOUBLE;**

- PROC
- print
- data=TEST
- SPLIT=* DOUBLE

The PROC key word is a required feature of the statement and is automatically taken care of within the SAS System. The procedure name is also a required feature, but it is up to the programmer to determine the name of the procedure. Data set references, parameters, and options are not required, and their presence is determined by the design.

Following the PROC statement are statements specific to the particular procedure invoked. There are, however, certain statements commonly used by many procedures. These statements are referred to as "standard statements" because they always have the same meaning and syntax regardless of which procedure they appear in. The standard procedure statements are:

- VAR
- BY
- ID
- CLASS
- FREQ
- WEIGHT

Aside from any standard statements which your procedure may use, you may also need new statements specific to the processing of your procedure. For example, the PRINT procedure has a PAGEBY statement which does not appear in any other procedure within the SAS System. Note that there are some statements which appear in many SAS procedures, such as the OUTPUT and MODEL statements, which are not considered "standard" statements. This is because the statements vary in syntax and/or meaning depending upon the procedure in which they are used.

The formal way in which procedure syntax is specified is called a "grammar." In general, a grammar is a set of rules which define the form (syntax) and meaning (semantics) of a language. Thus, in writing a procedure grammar, you will be defining the syntax and semantics of the statements to be used within your procedure. For the purpose of writing a grammar, a new base SAS procedure has been developed -- PROC GRAMMAR. The following statements comprise the GRAMMAR procedure:

**PROC GRAMMAR;**

- PROCNAME ....
- PROCOPT ....
- PROCParm ....
- PROCDATA ....
- STDSTMT ....
- LISTSTMT ....
- EQSMT ....

As the statement names imply, PROC GRAMMAR gives you the capability of naming your procedure, as well as defining options, parameters, data set references, standard statements, and new statements for your
procedure. Keep in mind that the input to PROC GRAMMAR is not itself a grammar, instead, PROC GRAMMAR is simply a tool used to generate a grammar. The output from PROC GRAMMAR will be the formal grammar specification for your procedure. For example, the following input to PROC GRAMMAR will produce the formal grammar for the PRINT procedure:

PROC GRAMMAR:
PROCNAME PRINT;
PROCINFO SPLIT (QUOTE);
PROCINFO DATA (INPUT);
LISTSTMT PAGE BY (
SUM (NIXVAR),
SUM (NDTVARS),
SUMY (MIXVAR))
END PROCEDURE;

See Figure 1 to see what the grammar generated by PROC GRAMMAR for the PRINT procedure looks like.

The words which begin with @ are called "semantic actions." Their purpose is to associate meaning to the syntax of the procedure. In most cases, it will not be necessary for you to understand the details of the generated grammar.

Although PROC GRAMMAR will generate a formal grammar for you, there are some limitations to the types of new statements which can be specified. The types of new statements which are possible are ones in which the statement name is followed by a list -- variable list, numeric list, etc. New statements which are not possible are ones in which the statement name is followed by a more complicated expression, like a mathematical formula. If your procedure requires statements which cannot be generated by PROC GRAMMAR, you will have to write the formal grammar yourself. Writing a formal grammar requires a great deal more understanding of semantic actions and of the process through which the SAS System goes to parse SAS statements. Difficult as it may be, the information necessary for writing a formal grammar is available in the SAS Programmer's Guide, and more complicated procedures are certainly possible. It might be interesting to note that 50 percent of the SAS procedures in base SAS software (including the statistical procedures), SAS/GRAPH, SAS/OR, and SAS/ETS have a syntax which can be specified within the bounds of PROC GRAMMAR.

Step 3: PARSER GENERATOR PROCESSING

After writing the grammar for your procedure (whether you write it yourself or generate it through PROC GRAMMAR), the next step is to process the grammar with the Parser Generator. Processing a grammar is similar in function to compiling a program -- taking code which is understandable to the programmer and generating code which is understandable to the computer. Along the same lines, the Parser Generator is like a compiler in that it is a piece of software designed to accept input in one form and convert it to another form. The input to the Parser Generator is a formal grammar; the output from the Parser Generator is a set of syntax rules in the form of a PL/I function. This function will be responsible for checking the syntax of the user's input to your procedure. Syntax checking is also referred to as parsing. Thus, the term Parser Generator is given to the software which generates the PL/I parsing function.

If you use PROC GRAMMAR to generate the grammar for your procedure, the Parser Generator processing will begin automatically at the end of the PROC GRAMMAR step. The Parser Generator is a stand-alone program not integrated with the SAS System. Therefore, the processing is done in a batch job that PROC GRAMMAR submits for you.

If you write your own grammar for your procedure, you will also need to submit the batch job which does the Parser Generator processing since it will not automatically be submitted for you. Once the batch job is complete, you will need to verify that the grammar you wrote has no errors. Just as a compiler may find errors with a program, the Parser Generator may find errors within your grammar. Error-free grammars are another advantage to using PROC GRAMMAR as a tool. Since the formal grammars generated through PROC GRAMMAR are in the correct form for Parser Generator input, it is not necessary to check for errors after Parser Generator processing is complete.

Step 4: SUPPORT ROUTINES

After the preliminary work of writing and processing the grammar for your procedure, the next step is to write the programs which support your procedure. These routines are written in a high-level programming language and should be compiled in the usual manner for your particular system. In general, these support routines can be broken down by function into four subroutines:

1. driver routine
2. parsing routine
3. set-up routine
4. analysis/output routine.

Keep in mind that the code you write as support routines is what will be executing when a user invokes your procedure within the SAS System. The SAS Supervisor is responsible for the first two steps of running your procedure:

1. Recognizing the PROC key word which will signal the beginning of a procedure step.
2. Recognizing the name of your procedure and calling the dynamic loader to load your procedure into memory and begin execution of your code.
After your procedure is loaded, control is turned over to your procedure for the duration of the current procedure step -- this is when your driver routine takes over. The driver routine is also referred to as the "proc seed" since it is where work for the procedure begins. For the purpose of dynamic loading and linking with the SAS System, your proc seed must have the same name as the procedure name used on the PROC statement. This name must not exceed eight (8) characters.

Driver Routine:

The driver routine is responsible for driving the procedure's execution. The first task of any procedure is initialization. The SAS Supervisor maintains several data structures which contain information your procedure will use later in its processing. These data structures must be initialized with a call to the XINIT interface routine. After initialization, the driver can begin its processing. Generally, the driver simply calls the other support routines to carry out procedure processing. Therefore, the typical flow of control within the driver is as follows:

- call XINIT
- call parsing routine
- call set-up routine
- call analysis/output routine.

Parsing Routine:

The parsing routine is responsible for checking the syntax of the user's input source and storing information related to the input. For example, the PRINT procedure has available on the PROC statement a SPLIT parameter. The parsing routine would verify that the keyword "SPLIT" was followed by an equal sign (=) and that the value assigned was a quoted string. The specific quoted string assigned to that parameter would also be stored in a data structure so that the procedure could later refer to it. Although the internal process of parsing is complicated, the work is done by the PL/I parsing function which was generated during Parser Generator processing (in conjunction with the SAS Supervisor). Therefore, the first role of your parsing support routine is to simply declare and invoke the PL/I parsing function. When you invoke this function, your procedure temporarily returns process control to the SAS Supervisor. The Supervisor will maintain control as long as there is user input to your procedure. The parsing mechanism within the Supervisor will check the syntax of the incoming source and store all related information in data structures. When process control returns to your parsing support routine, all information which was stored by the SAS Supervisor will be available to your procedure's code.

Notice that at the time that your parsing support routine regains control, all of the user's input for this procedure step has been seen by the SAS System -- from the PROC statement all the way through to the next RUN, PROC, or DATA statement. It is also very important to note that while the SAS Supervisor was parsing the user's input, each statement was examined to see what type of statement it might be:

- a statement specific to your procedure
- a global SAS statement, e.g. OPTIONS, TITLE, LIBNAME, etc.
- a step boundary statement -- DATA, PROC, RUN, or ENDSAS.

Statements which are not specific to your procedure are handled automatically by the Supervisor. Therefore, as a procedure writer, you will not have to be concerned with the syntax of these special types of statements. There are, however, a few SAS system options, such as LINESIZE and CENTER, which can be specified on the OPTIONS statement and which may affect the workings of your procedure. The user may change the value of these system options in the midst of the procedure step. This should not cause you any problems as long as you don't rely on the value of the system options until AFTER parsing is complete. For example, consider what would happen if, before parsing, the system line size is set to 132. Say you store that value in a variable for later use. Then you invoke the parsing function which gives control to the SAS Supervisor in order to parse the user's input. If the user's input source looks something like

```
PROC PRINT;
VAR X Y Z;
OPTIONS LINESIZE=80;
RUN;
```

then you will have an incorrect line size when printing output. If, on the other hand, you wait until after parsing is complete, then the correct system option values will be available. (See the SAS Programmer's Guide for information on how to obtain SAS system option values).

Once control returns to your parsing support routine (just after the PL/I function invocation), the next step is to see if the Supervisor found any syntax errors while parsing the user's input. This is done by checking the flag PROCERROR. The Supervisor will turn this flag on in the case of a syntax error. This error flag is just one piece of information which is stored in the data structures during the parsing phase.

All information related to the user's input source is stored in a data structure named STMTSTR (for "statement structure"). The following type of information is stored during the parsing phase:

- what options were specified on the PROC statement
- what parameters were specified on the PROC statement, and what value was assigned to each parameter.

691
what SAS data set references appeared on
the PROC statement

what standard procedure statements
appeared within the input source, and
what variable list was specified for each
statement

what new procedure statements appeared
within the input source, and what list(s)
(variable lists, numeric lists, name lists,
etc.) were specified for each statement.

The above information is stored in various
fashions. Options are data types which can be
either "on" or "off." Thus, options can be
expressed in a single bit of memory. If a
particular option is specified on the PROC
statement, that option is said to be "on," and its
.corresponding bit within the STMTSTR structure
is set to 1 (turned on). If an option is not
specified on the PROC statement, then it is said
to be "off," and its bit is set to 0 (turned off).
Parameters, on the other hand, are assigned
values by the user. The actual value assigned
to each parameter is part of the information
stored in the STMTSTR structure during the
.parsing phase. A data set reference is stored
as a pointer, or memory address, which indicates
its location. Lists can be stored in several ways
depending upon the nature of the items contained
in the list.

The overlay technique mentioned earlier is the
method used to extract information from the
STMTSTR structure in the appropriate format.
Aside from writing your own formal grammar,
overlaying is one of the most difficult concepts
to grasp in SAS procedure writing. Therefore,
don't get discouraged at the amount of time and
effort required to master this scheme. The
ultimate goal, at this point in your parsing
support routine, is to extract the input
information (option bits, parameter values, data
set pointers, etc.) and assign them to variables
declared within your support routines. By
assigning this information to variables in your
code, your support routines will have easier
access to the input information.

Thus, the general flow of control within a
.parsing support routine is:

- call parsing function
- extract user's input
  - options
  - parameters
  - data set references
  - lists

Set-Up Routine:

The set-up routine, as the name implies, is the
time to prepare for data analysis. The specific
tasks within the set-up routine will vary
depending upon the purpose of your procedure.
However, the general goal of this support
routine is to take the information gathered from
the user's input source and to use that
information to define the analysis environment:

- What SAS data set(s) will be used as
  input (if any), and what variables from
  that data set will be used in the data
  analysis process.

- What variables will result from the data
  analysis, and will their values be written
to an output SAS data set. If so, in
  what order will the variables appear in
  the output data set.

- How will the lists specified on standard
  and new procedure statements affect the
data analysis process.

The first and second items above are similar in
that they require you to make an association
between variables in a SAS data set and
variables declared in your support code. A
special association is required because SAS data
sets cannot be directly read from or written to
by your support routine. Instead, there are
special X-routines designed for that purpose.
When an X-routine, such as XVGET, reads from
an input data set, it will store the values from
the data set in variables which you specify. In
this manner, your variables act as "buffers" to
the SAS data set. Thus, by declaring buffer
variables within your support routine, and by
calling an X-routine to read the input SAS data
set, your support routines gain indirect access
to the values within the data set. This process
of reading a SAS data set and scattering its
values to buffer variables is called "scatter
read."

The inverse process, called "gather write," is the
means by which your support routines write
to an output SAS data set. Through calls to X-
routines, your set-up support code can define
exactly which buffer variables will be written to
the output data set. Then later, usually after
some data analysis has been performed, calls to
X-routines, such as XVPUT, will write the
values of the specified buffer variables into an
observation on the output data set.

Notice that the set-up routine is not the time to
actually read from or write to SAS data sets.
Instead, it is the place where the scatter read
and gather write variables are DEFINED through
calls to X-routines such as:

XVGETI -- initialize scatter read definition
XVGETD -- define scatter read variables

XVPUTI -- initialize gather write definition
XVPUTD -- define gather write variables.

Other tasks which are typically done at set-up
time are:

- Set up BY group processing if BY groups
  were specified.

- Initialize the print environment for output
  listings.

- Check for an empty input data set (0
  observations).
- Count the number of variables in the input data set.
- Read variable names and attributes from the input data set.
- Allocate needed memory for data analysis.

Analysis/Output Routine:
The analysis support routine is where the "work" for a procedure takes place. In many cases, the work will include reading observations from an input SAS data set and manipulating the data in some way. Other tasks which may be involved in data analysis are:
- Check for input values outside of a valid range.
- Check for missing values.
- Write new observations to an output SAS data set.
- Write analysis results to an output listing.

Like the set-up routine, the analysis/output routine will vary depending upon the function and goal of your SAS procedure. For example, in the analysis/output routine for PROC PRINT, observations are read from an input data set and printed to an output listing. In the case of PROC MORTGAGE, there is no input data set. Thus, the analysis/output routine for the MORTGAGE procedure simply calculates and prints tables.

Step 5: LINKING WITH SAS SYSTEM

The last step in writing a SAS procedure is linking your procedure's support code with the SAS System. In order to link successfully, the support routines for your procedure must be compiled without errors.

In general, the objective of this step is to create an executable image (load module) which the dynamic loader can locate and load into memory when your procedure is invoked within a SAS job. The details required for linking with the SAS system vary depending upon the machine and operating system which you are using. See the SAS Programmer's Guide appendix for your particular system for more detail.

DATA REPRESENTATION

Although the intent of this paper is not to present the technical details involved in writing a SAS procedure, some general statements should be made with respect to internal data representation.

The use of lists is an important concept in the SAS procedure environment. Each of the standard procedure statements requires a variable list to appear on the statement. During the parsing phase, the SAS Supervisor stores the variables specified by the user into a "list" data structure. However, it is not the actual names of the variables which are stored. Instead of variable names, the list contains integer values which represent each variable's relative position within the input SAS data set. In addition, there is a standard "LIST 0." List 0 contains the relative positions of all variables in the input data set. For example, if the input SAS data set contains six (6) variables in the following order:

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

then variable X would have a relative position of 1, variable Y would have a relative position of 2, and so on. Using the above input data set, the following SAS source code would result in the creation of two lists:

```
PROC PRINT;
VAR X  Z C;
RUN;
```

Lists are used throughout SAS procedure processing. Some lists, as in the example above, are created by the SAS Supervisor during the parsing phase. However, you may also create lists in your support routines through calls to X-routines.

Another important aspect of internal data representation is the manner in which information is stored in the STMTSTR structure during the parsing phase. Remember that during parsing, the SAS Supervisor stores information pertaining to options, parameters, data set references, and statement lists. Options, as mentioned before, are stored in single bits of memory which are either turned off or on. However, parameters, data set references, and lists have more than a single bit of information associated with them. Thus, they require a more complicated scheme of representation. When the Supervisor parses a parameter, data set reference, or list, two numerical codes are stored in the STMTSTR structure. The numerical codes are referred to as "type" and "mode." The numerical value of the type code identifies which of the three types of information is being stored -- parameter, data set reference, or list. The numerical value of the mode code further distinguishes the sort of information that is stored in the STMTSTR structure. For example, if a data set reference appears on the PROC statement, then the type code will be stored as 4. A type of 4 signifies the presence of a data set reference. Furthermore, if the data set specified is an input data set, then the mode code will be stored as 1.

693
A mode of 1 signifies that the data set was opened for input. Thus, there are various "modes" associated with each "type."

**TYPE:**

<table>
<thead>
<tr>
<th>TYPE</th>
<th>MODE</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 (Parameter)</td>
<td>1 (Numeric)</td>
</tr>
<tr>
<td></td>
<td>2 (Character)</td>
</tr>
<tr>
<td></td>
<td>6 (Format)</td>
</tr>
<tr>
<td>4 (Data set)</td>
<td>1 (Input)</td>
</tr>
<tr>
<td></td>
<td>2 (Output)</td>
</tr>
<tr>
<td></td>
<td>3 (Update)</td>
</tr>
<tr>
<td>3 (List)</td>
<td>11 (Numeric variable list)</td>
</tr>
<tr>
<td></td>
<td>12 (Character variable list)</td>
</tr>
<tr>
<td></td>
<td>13 (Mixed variable list)</td>
</tr>
</tbody>
</table>

This table represents only a portion of the possible modes for each type. The two important points to note here are:

1. The SAS Supervisor stores parsing information in numerical codes.
2. Your support routines can use the type and mode numeric codes in order to determine the format of the information which was stored by the Supervisor. In knowing the format of the information, your support routines can determine which memory overlay to use in order to extract that information.

**CONCLUSION**

It is not feasible to present all the necessary technical details within the confines of this paper. However, if you have developed a conceptual image of the interface between the SAS Supervisor, grammar specifications, and support routines, then you are well on your way to becoming a successful procedure writer.

As with all programming tasks, most of the learning and understanding does not take place until you actually begin writing the code. You will also find that after your first successful SAS procedure, subsequent attempts will seem far less complicated. Therefore, begin with a procedure which performs simple tasks. Then, after you have mastered the basics, work your way up to more complicated procedures.

If you feel that your SAS procedure has widespread application, don't hesitate to submit your work to be included in the SUGI Supplemental Library. This library contains user-written procedures, which are distributed as a supplement to the base SAS System software. If your procedure is included in the SUGI Supplemental Library, SAS users throughout the world will be able to benefit from the work you've done.