Design Concepts for SAS Applications
Ross Z. Merlin, Pinkerton Computer Consultants, Inc.

Introduction
This paper is for SAS programmers at all experience levels -- novice, intermediate and expert. Five general subjects will be discussed: Database Design, Programming Techniques, SAS Features, Utility Programs, and Resources for Programmers. Each section includes a discussion of the concept as well as the relevant features within the SAS language, references to papers by other authors, and references to the SAS User's Guide: Basics.

Database Design
A common problem encountered when designing a database for historical data (i.e. time series) is the choice between horizontal and vertical orientation. There are two choices: choose one orientation, forcing some users to use PROC TRANSPOSE every time they access the database; or keep two versions of the data, one in each orientation -- this may double storage costs and require twice as much processing to update.

<table>
<thead>
<tr>
<th>COUNTRY</th>
<th>YEAR</th>
<th>POP (vertical)</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>1980</td>
<td>205</td>
</tr>
<tr>
<td>000</td>
<td>1981</td>
<td>206</td>
</tr>
<tr>
<td>001</td>
<td>1982</td>
<td>208</td>
</tr>
<tr>
<td>001</td>
<td>1980</td>
<td>142</td>
</tr>
<tr>
<td>001</td>
<td>1981</td>
<td>142</td>
</tr>
<tr>
<td>001</td>
<td>1982</td>
<td>142</td>
</tr>
<tr>
<td>002</td>
<td>1980</td>
<td>167</td>
</tr>
<tr>
<td>002</td>
<td>1981</td>
<td>167</td>
</tr>
<tr>
<td>002</td>
<td>1982</td>
<td>166</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>COUNTRY VALBO VALBI VALB2 (horizontal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
</tr>
<tr>
<td>001</td>
</tr>
<tr>
<td>002</td>
</tr>
</tbody>
</table>

To make the best decision, one must analyze not only what reports are to come from the data, but also how those reports will be produced -- by PROCs or by DATA steps. PROCs generally favor a vertical orientation (e.g. a field called YEAR) which facilitates BY-group processing and column totals, whereas DATA steps generally favor horizontal, i.e. working across fields on the same observation (e.g. the year is indicated by the name or position of the repeating field) which facilitates the use of the SUM function and row totals.

Another database design issue involves improving access time to the desired records within large databases. If certain selections are very common, it might be advisable to split the database into several smaller databases, each unique by, for example, YEAR or STATE or PRODUCT. In some situations a more efficient approach would be to use the direct access technique with the POINT= option of the SET statement. This technique, which is not at all difficult to use, is discussed in the paper "Efficient Data Retrieval -- Direct Access Using the Point Option" by Neil Howard of ORI Inc. and Linda Williams Pickle of the National Cancer Institute. This paper appears elsewhere in these Proceedings.

When designing a new database or a new applications system, don’t use the old methodology of input -- processing -- output; instead, the order of consideration should be output -- input -- processing. To understand why, consider the sequence of steps used to plan a trip:

1) Decide where to go -- (output)
2) List the resources available (money, access to airport, railroad, or car) and any other constraints (fear of flying, departure times) -- (input)
3) Select the mode of transportation that best fits the constraints and determine the appropriate departure time -- (processing).

This approach corresponds more closely to the way (logical) humans think than does the IPO method:

1) I have $X and I want to get somewhere at mid-day Saturday
2) I can travel by airplane or train
3) Therefore I want to go to SUGI.

One of the benefits of the OIP method is that it helps eliminate useless fields from the data source, although one must think ahead to possible future uses for such fields.

This technique is discussed by J. Harvey Trimble Jr. of Trimble Data Systems in his SUGI '83 paper entitled "Structured Methods for SAS Applications", pages 103-106.

Programmers can take advantage of the self-documenting nature of SAS databases by keeping the following suggestions in mind:

1) Use relevant dataset and variable names.
2) When there are multiple datasets for a project, standardize names, types, lengths, and codes. For example, different datasets might refer to a country code by COUNTRY, CTY, CNTRY, CC, CTRYCODE, etc. Within these fields, different coding schemes might be used: FIPS, Agency...
for International Development, World Bank, IMF, the literal name (e.g. ZAIRE), etc. Choose the one or two codes most common within the data and make the others available through formats.

3) Use dataset labels to describe the dataset, refer to documentation, specify the name/dept./telephone of the person responsible for the data, the expected sort order, or the OS DSN of the jobstream which creates or updates the dataset.

4) Use variable labels to describe the variable, refer to documentation or a data dictionary entry, dimension units (e.g. furlongs per fortnight to measure speed), or to identify the source of the data (e.g. Drunken Wagonal's Encyclopedia).

5) Remember that some source statements are retained in the database and are available via PROC CONTENTS -- therefore, use plenty of comments and observe stylistic conventions.

To add to the self-documenting capabilities, each database should contain a SAS dataset called NOTES which should have two fields: DATASET, an eight byte character field to contain the names of the other datasets (or blank for general information); and NOTE, a 40 to 65 byte character field for the descriptive text. This provides greater flexibility than that afforded by dataset and variable labels, which are limited to 40 characters each. This is similar to the data dictionary concept discussed by Judith L. Cohen of Social and Scientific Systems Inc. (now of ORI Inc.) in her SUGI '83 paper entitled "Automatic Generation of Documentation for a Large Public Use SAS Database", pages 412-417.

Programming Style

For one opinion on programming style, refer to "Some Guidelines for Documenting SAS Source Code" by Scott L. McGregor and Mary Nelson of Hewlett Packard Co., SUGI '84 Proceedings, pages 119-122. Another opinion was presented in a SUGI '84 poster entitled "Improving Programmer Productivity With Coding Conventions" by Joshua Sharlin, Anne Asher, and Migdalen Eley of ORI Inc.

There is generally no such thing as a "one-shot" SAS program. There are three typical situations that give rise to such a statement:

1) You write an ad-hoc report for someone who tells you he/she needs it just once for a presentation to the boss -- a year later, the same person returns with "Remember that little report you did for me last year? The boss liked it so much he wants to see it again this year!"

2) There may be some technique that you use infrequently, such as direct access of SAS datasets or a particular SMF analysis. Rather than have to relearn or rediscover the technique when it is next needed, it is easier to simply review an existing well-written program.

3) Your "one-shot" program might be an excellent teaching example, or a good way to "show-off" how powerful SAS is.

The point of all this is that every program, no matter how trivial it may seem when it is created, should be written with full respect for style and documentation (i.e. comments).

Specific coding guidelines are well covered in the works cited above; for those who must see some "rules" now, consider the following:

1) Rule #1 is that all rules are meant to be broken. ALWAYS keep in mind that the objective of these coding rules is to have code that is more readable -- if a rule gets in the way of that goal, it is not doing what it is intended to do; so ignore it. Think of the 55 m.p.h. speed limit -- most of us agree with its goals of saving lives and conserving gasoline, but sometimes...

2) In general there should be only one statement per line except when the entire PROC or DATA step will fit on one line, e.g.: PROC PRINT; BY COUNTRY; RUN;

3) Use comment, SKIP, and PAGE statements, and blank lines to divide the program listing (log) into logical units.

4) End each DATA or PROC step with "RUN;". This is particularly useful when editing a long DATA step -- to find the start of the next step, you don't always know if you should look for "DATA" or "PROC"; but if every step ends with "RUN;", you have something consistent with which to search.

There are at least two ways to manage formats. The traditional approach is to maintain a format load module library. It is important to also maintain a format source library, and to ensure that each format load module has a corresponding source member and that the source is the same version that created that load module. All too often there are several versions of the source code and no one can remember which one goes with the format module. To preclude this situation, have one person responsible for copying members into these libraries, and restrict all
other users so that they have only READ access (e.g. RACF: UACC(READ)). The other approach is to store all the format values and labels as SAS data and maintain a utility program to generate PROC FORMAT code from the data. One advantage is that there is no source code to debug (it is all recorded from the data); another is that the format values are available to be attached to other datasets by a MERGE operation.

If you already have some format modules without their corresponding source code, the source code may be recreated by using PROC FMTLIB, which is documented in the SUPPLEMENTAL LIBRARY USER'S GUIDE.

If you make just a little extra effort when creating your datasets, PROC CONTENTS may provide a great deal of useful information about your data. In general, PROC CONTENTS can provide information at three levels: the OS dataset level, the SAS database (data library), and the SAS dataset level. At the OS dataset level, PROC CONTENTS displays information such as the OS dataset name, the name of the volume on which it reined, the DDF attributes, any information concerning its creation (USERID/USERID, CPU, date-time, etc.). At the SAS database level, PROC CONTENTS displays for each dataset within the database: its name, the number of observations, and how much space is used (the last two items are provided for disk datasets only). The most detailed information occurs at the SAS dataset level, where the information displayed includes: details about the dataset's creation, the dataset's label, each variable within it, and the source statements of the DATA step or the name of the PROC that created the dataset. For each variable, PROC CONTENTS displays its name, position, type, length, informat, format, and label -- the informat, format, and label information can be displayed only if you used the corresponding statements when the dataset was created; or, if they weren't used, SAS provides a second chance through PROC DATASETS.

Please take the time to read the PROC DATASETS section in the USER'S GUIDE -- there are many capabilities new to SAS82 of which you might not be aware.

Many programmers don't bother with INFORMAT, FORMAT, and LABEL statements in DATA steps for three reasons. First, INFORMAT statements do not directly replicate their counterparts in the INPUT statement -- only the format type is stored (e.g. if "INFORMAT COMMA.2;" is used, only "COMMA." is stored). Second, there is the matter of style -- many programmers prefer to code the INFORMATs and FORMATS in their INPUT and PUT statements; comments (if used at all) are often used instead of LABEL statements.

Third, a false sense of security may be fostered by the generations of source code displayed by PROC CONTENTS. By default, PROC CONTENTS shows the five most recent generations of source code; if a dataset is updated or sorted occasionally, in a very short time the original source code is lost to five sets of "Data generated by PROC SORT" or other relatively meaningless information. Fortunately, the information from the INFORMAT, FORMAT, and LABEL statements is maintained independently of the source code and therefore does not vanish.

SAS Features

SAS provides many options to give the programmer the ability to tailor SAS to specific situations, but the price that is paid for this flexibility is a list of options and features that seems overwhelming to even experienced SAS programmers. Here are some of the most useful features.

The ? and ?? format modifiers (pages 85 and 86 of Basics) are useful when you anticipate the possibility of "dirty data". The ? suppresses the "Invalid Data..." note normally printed in the log; the ?? also suppresses the dump of the input lines. When SAS encounters an error in the input data, it prints an error message (unless ?? was used) each time that error occurs, up to the number of times specified by the ERRORS= option. The default is twenty. Usually you can decide that you have a problem with fewer than twenty different pieces of evidence -- specify OPTIONS ERRORS=1 or whatever number you feel is appropriate. ERRORS= is discussed on pages 331 and 480 of Basics.

When debugging a SAS job, often it would be useful to save all of the temporary (WORK) SAS datasets and examine or manipulate them as needed. One way would be to change each single-level name to a two-level name, where the added (i.e. left-most) level refers to a DDNAME allocated to a "permanent" SAS data library. After debugging, it would be necessary to undo all the name changes. Don't despair, there is an easier way. Simply use "USER" as the DDNAME for
the data library where you want the WORK datasets saved. That is all that need be done -- simply have DDNAME "USER" present and SAS will name all the temporary datasets as USER.whatever instead of WORK.whatever. After the problem has been solved, simply remove DDNAME "USER" and the temporary datasets will once again be prefixed with "WORK". If a more relevant name than "USER" is desired, specify that name in the option USER=. The default is USER=USER. For more information, consult pages 350, 364-368, and 875 of Basics. This approach is better than allocating your WORK DDNAME to a permanent dataset -- very strange things can happen when you reuse a saved WORK library, since SAS stores some control information there. Normally this control information is transparent to the user and is only apparent when certain errors occur (such as running out of space in a data library) or if the MAP keyword is specified with PROC CONTENTS.

If you are executing SAS interactively and plan to use PROC PRINTTO to send printed results to a printer or file other than the terminal, you will probably specify "OPTIONS PS=60 LS=132 CENTER OVP;" -- and when you want results back on the terminal, you will have to undo all of those options. Since what you are effectively doing is using the defaults for BATCH and then the defaults for TSO, you can simply specify "OPTIONS DEFAULT=BATCH;" or "OPTIONS DEFAULT=TSO;" directly. This is documented on page 329 of Basics, and despite what the book says, "BATCH" defaults for TSO, you can simply specify "OPTIONS DEFAULT=TSO;" or "OPTIONS DEFAULT=BATCH;" directly. This is documented on page 329 of Basics, and despite what the book says, "BATCH" seems to work fine under MVS. When using this feature, keep in mind that switching option group resets the PRINT file to its default (FT12F001); therefore specify PROC PRINTTO after DEFAULT=BATCH is used. Also, one of the BATCH defaults is usually "NOSPOOL;" generally specified as OPTIONS DEFAULT=BATCH. NOSPOOL;" provides optimal results.

If you have not allocated the proper format library or run PROC FORMAT in your job or session, and you are using a SAS dataset that has a format permanently associated with a variable, an error message will appear every time that dataset is referenced. One way to avoid this problem is to create a dummy format: "PROC FORMAT; VALUE "WHATEVER; RUN;". An easier solution is to specify "OPTIONS NOMFERR;", which tells SAS not to consider a missing format to be an error (however, a NOTE is still printed in the log). For further information, consult Basics page 333. Similarly, there is an option NODSNFERR which tells SAS not to consider a "dataset not found" situation as an error (SAS proceeds as if an NULL dataset had been specified). This option is discussed on page 329 of Basics.

SAS permits execution of programs written in other languages (e.g. FORTRAN, COBOL, PL/I, ASSEMBLER) from within a SAS job. To use this capability, make the library containing the load module available to SAS by allocating it to DDNAME SASLIB (OS BATCH or TSO) or by specifying the name of the library in the LOAD parameter of the SAS TSO command; when the program is to be executed, specify the member name of the program load module on the PROC statement, e.g. "PROC WHATEVER;" (the member name must not be identical to any name already in SAS.LIBRARY, nor the same as any FORMAT). Communication of data between SAS and the external program is easily handled by using sequentially accessed files; parameter information can also be sent to the external program via the PARM= option. See Basics page 258 for details.

Utility Programs

In many applications systems there often arises the situation whereby one program, perhaps with slight modifications, could be used to satisfy many requests. Usually these are many programs of this type, often for report generation, data extraction, etc. The slight modifications are easily handled by CLISTS and the new MACRO language; the tricky part is making these programs available to all who might use them. Here is a method to make such programs readily accessible.

1) Create all such programs as new-style macros with no parameters (handle parameters by %PUT and %INPUT statements).
2) Store these programs as members in a macro library (MACLIB).
3) Create "HELP" members for all of these macros, either as new members in SASHELP or in a similar library concatenated to the SASHELP DD.
4) Create a driver macro which:
   a) accepts a macro name (member name) as its only parameter
   b) validates the member name
   c) checks to see if that macro has been used in this session -- if not, the member is %INCLUDED from MACLIB
   d) executes the macro.
5) Create a HELP macro to describe the driver and which macros are available.
6) Create an alternate version of the SAS command to enable these
macros. The command allocates the MACLIB, specifies \texttt{OPTION IMPLMAC}, and executes an \texttt{INITSTMT} (\textit{Basics} page 335) such that the "driver" is automatically loaded, e.g.

\begin{verbatim}
... OPTIONS='&OPTIONS SORT=ASORT IMPLMAC
INITSTMT=\'\'\'\'&ZINC MACLIB(DRIVER)\'\'\'\'
\end{verbatim}

To see how this works, assume that the XYZ division has their version of this method. When they want to use SAS, they type \texttt{SASXYZ}. When SAS comes up, they can execute normal SAS statements, or they can issue their special XYZ commands, such as \texttt{XYZ REPORT7;}. Since IMPLMAC was specified, SAS recognizes "XYZ" as a macro command (this is the "driver"), and "REPORT7" is its parameter. The "driver" executes macro \texttt{REPORT7} (loading it only if necessary). These macro commands can be viewed as a language in their own right, allowing users who don't know any SAS at all to take advantage of existing SAS programs.

Copies of the code for the method described above are available from the author.

Resources for Programmers

In addition to the SAS user's guides, the Applications Guide, and the technical reports, there are many other sources of SAS-related information:

1) Previous editions of \textit{SUGI} Proceedings.

2) Every installation should use the NEWS feature to advise SAS users of the availability of new releases, new documentation, meetings of user groups, changes to defaults, etc. News text is updated simply by editing the appropriate member in SAS.HELP. Installations are free to add members to the SAS.HELP library, or to concatenate a similar library to the SASHELP.DD in the JCL procedure and the CLIST.

3) The SAS installation tape includes a sample library of SAS programs (SAS.SAMPLE), and also includes instructions on how to produce an index to the sample library. See to it that the library is accessible on-line, and periodically browse the members.

4) Be sure that your name is on the SAS Communications mailing list (the free quarterly newsletter from SAS Institute), and make sure that other SAS users have timely access to new issues.

Join (or start!) a local SAS User's Group, and share your ideas with your fellow SAS users.

\textbf{NOTES}

SAS and SAS Communications are the registered trademarks of SAS Institute Inc., Cary, NC, USA.


The author welcomes comments and questions:

Ross Z. Merlin
P.O. Box 6520
Silver Spring, MD
20906-0520
(202) 532-0032