Scott L. McGregor and Mary Nelson, Hewlett-Packard Company

Abstract
SAS provides a powerful implementation language for systems which are designed to be extensible, that is, where the users may extend and customize the capabilities of the systems by writing and then executing new functions. SAS is easy to master and can handle high-level constructs. This makes SAS ideal for writing and executing extensions in real-time. SAS 82 will further enhance this capability. This paper will investigate some areas where the extensible features of SAS may be put to greatest benefit. Suggestions for additional enhancements to SAS to further facilitate these kinds of systems will be made.

In particular, the paper will focus on the use of SAS in (1) an extensible editor, (2) a dynamic graphics program, (3) an operations research system, and (4) an automated documentation generator. This paper assumes the existence of SAS, SAS/GRAPH and SAS/PSP in the working environment.

Introduction
SAS is a powerful language for many applications. The following paper will discuss some areas where SAS is typically not used, and which aspects of SAS suggest that it might be useful. In some cases this would require modifications to SAS. However, many of these modifications are within the realm of a competent programming group, and would not require SAS Institute support. These could be distributed via the supplemental library. Of course, high user interest may induce SAS Institute to make such additions standard to SAS. In any case, during the course of this paper we hope to suggest some new ways of looking at and using the inherent power of SAS.

The authors base this analysis on experience gained from the design and implementation of several systems written in SAS at the Hewlett-Packard Corporate Data Center. However, the thrust of this paper is visionary, and past experience can provide only a base for the projections contained here. There are many other valuable applications areas where SAS might be enhanced, those discussed below represent only a subset of the possibilities we have explored.

An Extensible Text Editor
One of the great powers of SAS is its ability to create SAS source code which can then be inserted into the currently executing code. This is implemented via the %INCLUDE feature. This capability is of particular value in an extensible application. An example is a text editor, which allows the creation of new edit commands in the midst of an editing session, and immediate application of these commands to the text being edited. This can simplify the processing of tedious changes.

SAS is an ideal language for this, because its syntax is very simple and easy to use. If we imagine that the text being edited is in a SAS database similar to those created by PROC FSLETTER, we can see that it should be easy to write SAS DATA steps that could operate on the text to provide text justification and other complex formatting commands. If these DATA steps were assigned to macros, and parameters were passed using the new SAS 82 features, the user could execute them by a simple invocation of the macro name. Frequently used text transformations could be coded as SAS PROCs or functions and addressed by their names.

This approach currently may be used in any interactive session. However, to make it more friendly, it would be desirable to package it in a screen editor interface (such as FSLETTER) and allow direct modification of the database so that multiple modifications do not require separate data steps. This would require user or SAS Institute modifications to SAS. What is needed is a way to invoke PROCs, functions, and DATA step macros against the text database from within, say, PROC FSLETTER. Allowing names and parameters to be typed into the command line would cause execution. If this is not an already defined FSLETTER command, the PROCs, functions and macro names would be checked, and if a match is found, executed. Additionally the user needs to be able to switch to code writing mode without losing position in the text, so that new macros can be designed and run on the fly. Additionally, it would be desirable to allow commonly used macro names, etc., to be associated with the user's free PF keys.

What would this enable a user to do, that cannot be done already with an existing editor such as the TSO editor, the SPF editor or FSLETTER? A simple example can demonstrate the increased power of such an editor. Let us suppose that our text file in the database contains job JCL. Furthermore, let us suppose that this JCL contains numerous references to disk
datasets containing the index BKUP. We would like to change residency to tape. For datasets being created, we not only want to change from DISK to TAPE, but we want to eliminate any associated SPACE parameters. Since the SPACE parameters may be on separate lines from the UNIT parameters and from the DSNAME parameters, we cannot do this in any straightforward way in most text editors. However, in our extensible SAS editor, we could write a simple data step that does the following:

1. Read JCL lines into an array until you get to one that doesn't end in a comma (the comma indicates that the statement is continued on the next line).
2. Search each line in the array for a DSNAME parameter.
3. Check whether the DSNAME parameter contains a BKUP index.
4. If it doesn't, output the lines unchanged.
5. If it does, check the UNIT parameter.
6. If the parameter is TAPE, output the lines unchanged.
7. If the parameter is DISK, change the parameter to TAPE.
8. Check for a SPACE parameter. If not found, output lines.
9. If the SPACE parameter is found, delete it.
10. Check to see that the final line is not a null continuation. If it is, delete it and the preceding line's trailing comma. Output the lines.
11. If the final line is not a null continuation, output the lines.

Having coded this DATA step on the fly and assigned it a macro name we could invoke it on our current file and make the changes in one simple command.

**Dynamic Graphics**

With more graphics terminals becoming available, the such terminals are becoming a staple at many data centers. SAS/GRAPH provides a good tool for utilizing many of the capabilities of these devices, but its utility is limited when refining graphics. In particular, there is no easy way to create and modify such common diagrams as flowcharts and circuit diagrams. The creation of special fonts with such special symbols as common flowchart and circuit symbols would simplify this work, but such designs usually require iterative refinements, and SAS/GRAPH does not currently provide these.

What's needed is an user interface such as PSEdit, where draw and move commands, as well as text commands can be entered and then drawn. If they are not correct, they could be easily changed. We have experimented with such an interface using SFF edit and SAS %INCLUDE statements and found it helpful, but a more complete interface is needed to make such a facility truly usable.

If such a system were developed in a very dynamic user environment such as described above (an extensible text editor), SAS/GRAPH could be useful in doing computer graphics animation as well.

**Operations Research**

The addition of some simple operations research PROCs could provide SAS with some additional capabilities of use to many users. With such PROCs, management science problems such as scheduling could be solved using PERT-CPM, and optimization problems could be solved using Linear Programming and Dynamic Programming. Algorithms for each of these applications have been published by groups such as the Association for Computing Machinery (ACM), and are widely available.

Using locally written PROCs, we have used PERT analysis to do sophisticated project management and to build and analyze job dependency trees for managing our schedule of production jobs. SAS provides a good environment for preparing the data for such analyzers and as an excellent environment for preparing and displaying results. Some additions to SAS/GRAPH which would make the displaying of PERT networks simpler would also greatly enhance such capabilities.

Again using locally written PROCs, a linear programming program was made available within SAS for doing capacity planning. SAS simplified massaging of both input and output, and via PROC MATRIX, extended analytical capabilities greatly. While the use of linear programming to solve such problems as optimizing production and employment scheduling is well known, these techniques can also be used effectively in solving many common problems encountered in the day to day running of a large data center. They also may be applied in many unexpected areas; for example, Donald Knuth (1) has discussed how dynamic programming methods can be used to effectively justify text in variable width character fonts.

While existing Operations Research programs may be simply called as PROCs without modifications, to make such PROCs easy to use for most SAS users, special PROCs are needed to handle parameter passing and allow data to be read from SAS databases.
Automated Documentation Generators

In a paper presented at the International Conference on Systems Documentation, January 22-23 presented by the ACM SIGDOC and SIGOA groups, Christopher Hartsough et al. (2) of the Jet Propulsion Laboratory discussed a system "for the generation of typeset quality documentation from a formal database." Their system "directly supports the conceptual separation of system design, document content design, and document format design".

By using a structured database, they were able to describe in detail a complete system design. Using a program that read an "outline" or document template database they were then able to produce actual text documentation of the system design. This was then fed to a "pretty printer" program to produce typeset quality output. While the authors of the above paper did not use SAS to store their system design description, text content description or print format description, a SAS implementation could conceptually simplify the implementation of such a system.

The mechanics of such a system are simple in theory. Anyone who has ever received a personalized form letter has a model of how such a system works. A general template definition of the output document is generated and the variable elements from the systems design are placed into predefined areas in the template to produce the final document. Actually, a system such as the one defined in the above mentioned paper, is much more complex and can create highly sophisticated and individualized documents, reflecting the depth of analysis done on the template by the original techwriter.

PROCs which allow the user to define the system design, document content design and document format design could be created to simplify the definition of these databases. FSEDIT would provide a data entry vehicle. A PROC could generate text from the system design database using the text content database. Another PROC could invoke the "pretty printer" to prepare the final output.

Such a system may well require the most effort of any of the applications described in this paper. However, the use of SAS could simplify the design of such a system, and provide an easy-to-maintain and evolve system language. Development of such a system, using directions given by Hartsough et al., is well within the capability of a large data center and probably worth the effort for any group doing a large amount of systems documentation.

Summary

This paper has attempted to show SAS's capabilities make it a good choice for developing systems in a variety of areas where it has not traditionally been used, including extensible editors, dynamic graphics, operations research, and automated documentation generators. While these areas are diverse, SAS's high level constructs and run time compilation make it a versatile tool that is well suited to the needs of software developers in these areas.

References:
