A SAS program can be an effective planning tool for a manager with little or no computer training. If a SAS programmer gives a manager a completed SAS program, a cataloged JCL procedure, a set of fill-in-the-blanks macros, and an input data set, the manager will then have a complete Decision Support System. The macros and the data files allow the manager to play "what-if" games and test different forecasts and different sets of business conditions or different allocation bases. PROC MATRIX, combined with a programmer-formatted report, facilitates spreadsheet formats and thus presents the manager with the information in the form to which he or she is accustomed. Furthermore, if the user has a macro which allows a variable number or repeated calls to a macro within the program, he or she can use an iterative algorithm and test the amount of iterations necessary. Finally, year-to-date totals and counters can be stored in SAS data sets in generation data groups, thus increasing the efficiency of regularly run programs.

**INTRODUCTION**

In many large organizations, the people who need to use the computer are not the same as those who can use the computer. Those who need to use the computer are very often people in middle or upper management whose responsibilities include some form of planning. They may need to allocate work to different plants; they may need to set hiring or marketing goals; they may need to prepare budgets and financial statements. These people are usually on the distribution list for a large variety of computer-generated reports, but they receive them after the fact and in a pre-determined format. The reports are very often unable to change the nature of the reports, to get the information formatted, selected, or totalled differently, or to ask what-if type questions regarding the reports. When they make such a request, they may encounter a veritable Data Tower of Babel and hear responses such as:

- "Too many files need to be cross-referenced."
- "That will take two years."
- "Talk to my project leader's system engineer."
- "That's written in BASIC and we're a PL/I shop."
- "That type of program is not within the scope of my responsibility."

Even when the manager knows a little bit about programming, he or she may not be allowed to insert their changes into the operating schedule. One of the following results usually occurs:

1) The manager does the best possible under the circumstances and continues without the desired information.

2) The manager gets on the data center's waiting list and gets what he or she desires six months after the requirements have changed.

3) Somebody writes a "bootleg" program for the manager in an undocumented, ad hoc fashion and the manager grows dependent on the undecipherable code.

This paper illustrates the use of SAS programs as a means of gaining the benefits of a bootleg program while having clear, flexible, and modifiable source code. When given one or more SAS programs, the manager can use the computer as a planning tool to quickly evaluate various strategies and contingencies and produce a nicely formatted report.

**SAS CAN HELP**

SAS is a very easy programming language to learn because of its structure. It performs a wide range of statistical, data-handling, and report-writing functions with a small amount of commands. Routine, repetitive programming functions are built into SAS as defaults. As the SAS user grows more sophisticated, he or she may override specific default values to tailor the reports and calculations more closely to their needs. Figure 1 shows an example of how a report may be progressively fine-tuned. Furthermore, many of the PROC's contain data handling and report-writing features.

Thus, by using a SAS program, someone can quickly generate a report in the format to which he or she is accustomed. Unlike in other very high-level languages, the user is not tied into a specific output format. An example later in this paper will illustrate exactly how PROC MATRIX can facilitate spreadsheet-type report formats.

Besides being easy to learn, SAS is very easy to make readable. It's free-form and not bound by column and line specifications, so the commands can be written in a readable manner. The recently added ability to use the PL/I-type command /* and */ facilitates documentation.

The use of MACRO, % and */ commands also helps to block out certain sections of the program which may only be needed intermittently. This saves the programmer and user from having to store and choose between many different versions of what is essentially the same program.

The defaults contained in the structure of SAS aid in speedy problem-solving. A moderately experienced SAS user is able to write a needed program very rapidly. Once he or she has mentally organized the program, it is merely a matter of choosing from the available modules. This lends itself very nicely to prototyping, or "middle-out" design, in which later decisions...
In a project are made as the results are received from the earlier experiments.

When receiving data from a large variety of sources one can use the missing value feature of SAS to verify the integrity and completeness of the data. At the same time, the program will continue to process good data. Figure 2 shows some examples of the error-processing features contained in SAS DATA steps. Use of the FIRST. and LAST. commands facilitates the identification of both major and minor control breaks. The processing can then be performed with an IF....THEN DO....END sequence.

USER-ORIENTED PROGRAMMING TECHNIQUES

Many middle and upper level managers have a continuing need to evaluate the results of various sets of conditions. Examples include a managerial accountant evaluating different allocation bases, a production manager trying to determine the effects of different product mixes, a personnel administrator planning a recruiting drive, and a marketing manager allocating scarce dollars. These people have various options and they are very knowledgeable about the results of their decisions on a one-step-at-a-time basis. However, their decisions usually set off a chain of events which is so complex that it takes up an unaffordable amount of their scarce time. They find themselves and their subordinates "number-crunching" when they could better use their time planning. The real world will not wait for their number-crunching and thus they are able to evaluate only a small subset of the available alternatives.

A single SAS program, or a collection of SAS programs, can be a useful solution to this problem.

There are many techniques which can be very useful in maintaining the flexibility of the SAS program so that the manager can continue to use it as his or her needs change.

CODED INPUT CARDS

Using different card-types as input to the program can be very helpful, if the user needs to supply some or all of the data in logically different forms. The user will organize the data in sensible groups, and then enter it onto cards or video card images whose first few columns contain the card-code (see Figure 3). If the people are filling the information in at a video with full screen edit, the programmer can provide the user with column headings which the program will ignore by using the FIRSTOBS option. (See Figure 4.)

MACROS IN THE JCL

In the case where the user is providing some of the program logic as well as input data, the use of macros in the Job Control Language (JCL) can give the user options concerning the execution of the program. Figure 5 shows a case where the title of the report is left to the user and Figure 6 shows an example which lets the user select both the information and the control fields.

For those who may be unfamiliar with JCL, Figures 5 and 6 show the macros being concatenated to the program by having the data-set SYSIN contain both the cards with the macros and the source program. The user is given a deck of cards like the ones in the illustrations and can control important aspects of the program logic without having to deal with the program itself.

VARIABLE NUMBER OF ITERATIONS OF AN ALGORITHM

In this case, there is a well-defined iterative algorithm which must be performed a variable number of times depending on a changing set of circumstances. A fair amount of routine processing takes place both before and after the algorithm. A solution is to have a macro contained in the program to perform the algorithm and to have a command immediately after the macro which invokes a user-controlled macro.

Figures 7 and 8 show an example of this type of program. The program allocates permanent investment from manufacturing to marketing businesses in a vertically integrated company. Figure 7 contains a flowchart of the program logic and Figure 8 illustrates the JCL and the macro in the program. Many of the divisions in the company sell both to outside customers and other divisions and many of the divisions both buy and sell internally, so it is often necessary to iterate the macro a different amount of times from the previous run. Since part of the report tells the user how much money was allocated on the last iteration, he or she can decide whether it is necessary to call the macro another time. This is easily done by inserting a card with the word "ALLOCATE" keypunched on it.

GRAPHS

The use of computer-generated graphs continues to increase as both hardware and software vendors make graphs easier to produce. Graphs highlight trends and illustrate important points quickly and easily. As the cost of a manager's time continues to grow and as the amount of information which managers must deal with continues to increase, the ability of graphs to present information more rapidly is valuable, especially when compared with tables of numbers.

SAS GRAPH is very helpful because of its ability to perform a wide variety of functions with the prompting of a small number of commands. Further, PROC GSLIDE can be used to create attractive introductory pages and transparencies. The examples which I am using here are from a market survey for a particular product line. In one of the examples using the block chart feature, market trends and market share can be plotted simultaneously. At the same time, the blocks can be color-coded to highlight whichever aspect is more important. As Figure 9 illustrates, once the SAS data set has been created, the manager can re-format each graph to suit his or her preferences.

333
needs. Since it only takes a few program statements to produce the graph, it can be changed on an as-needed basis. Furthermore, the information can be summarized or broken down into subcategories and then graphed whenever the need occurs. As in the earlier example, the decision on control breaks can be left almost entirely to the user by leaving the BY statement up to the user.

**SPREAD-SHEET FORMATS**

Many organizations have their own well-defined format for presenting internal financial information. Usually this is in a spreadsheet layout with each line representing some type of revenue or expense and each column representing a product line or subcategory. In addition, there are usually rows or columns which represent totals and percentages. Each page then represents a specific unit of the business and certain pages can be used to present an overall business summary.

Furthermore, the manager responsible for preparing these reports usually is given only the total figures and is responsible for allocating these totals to different geographical units or product lines and then re-summarizing to produce a consolidated financial statement. Although he or she may wish to try many different allocation bases for the different categories of revenues and expenses, the limited time allowed for the preparation of this financial statement will not allow such WHAT-IF games. Instead the manager uses his other instincts, makes "seat of the pants" decisions, and is forced to go with his or her first judgement.

A SAS program can change all of this. By using coded option cards, the user can call the SAS program, input different allocation methods, and test the results. Only four types of cards are necessary:

1. The total amount to be allocated in a particular category.
2. Different basis amounts for each product line which might receive an allocation. (The SAS program will translate these into percentages.)
3. Which allocation basis is desired for each line item.
4. Any lump sums which need to be transferred before the allocation is made.

Figure 10 shows a possible string of JCL which can be given to the manager who is using this program. Once the allocations are made, the program uses PROC MATRIX, as in Figure 11, to format the information and calculate percentages and totals.

When the manager sees the financial statements resulting from the allocations, he or she can change a small number of cards, try different allocations, and rediscuss the program. This takes the "number-crunching" away from the manager and gives it to the computer, allowing the manager to use scarce time more creatively and beneficially.

**INTERFACE WITH OTHER LANGUAGES**

There are many very high-level languages on the market right now which perform a variety of specific business tasks very well. However, they are often constructed in such a manner that, unlike when using SAS, it is extremely difficult, and sometimes impossible, to present information to them in a manner other than the default. It is also sometimes impossible to present the results of their calculations in a manner other than that built into the package. In these cases, SAS programs can provide valuable add-ons to these packages. The user can be presented with a unified package in which a SAS program will transparently translate his or her input into the form necessary for these languages and another SAS program will transparently translate the package's output into a familiar readable form.

In EMPIRE or IFPS the program requires a matrix with a fixed number of rows and columns and the Foreign File Interface can be difficult to use. Putting a SAS program at the front end of an EMPIRE or IFPS program allows the manager to send a variable amount of row, column, and scalar data and which is at different levels of summarization. The SAS program will summarize and format the data for entry into the EMPIRE or IFPS program.

In addition to being able to process various amounts and types of data, SAS can read the data in various formats and from different files. It may be necessary to merge information from different files together before the financial analysis is performed. Not all of the information needs to be user-defined. Much of it may already exist in computer-readable form on disk or tape. It may be character-form, packed, zoned decimal, or binary. SAS can read it in any form and then pass it on to EMPIRE or IFPS in a form acceptable to those languages.

MPSX is another software package which is very useful; in this case to a production manager who is attempting to allocate capacity. It performs linear programming on the given constraints and uses a profit function for optimizing. When there is not a feasible solution, MPSX reports the condition and gives the infeasible solutions which come closest to solving the problem.

But a person must have a fairly good knowledge of applied math and linear programming in order to use MPSX. He or she must understand how to set up the simultaneous equations and how to read the output from MPSX, which is in a fixed format. The production manager, who is the actual end user of the MPSX calculations, may not have such training and it may not be in his or her budget to have a person with that training on the staff.
What the production manager really needs to be able to do is to simply list his or her capacities, costs, forecasted loads, and preferences and then receive a report showing which machines to use, when to use them, and whether or not absolute limits are being bumped into. And this is exactly where the SAS programs are useful. One SAS program can read and merge the different files containing the production capacities and economics. It can then intercept anomalies and missing data and produce an error report, something which MPSX does sketchily, when it does it all. After the errors in the data are cleaned up, the SAS program can be re-run to format the data the way the MPSX program wants to see it. After the MPSX program runs, another SAS program can read the output, interpret it, and format it the way the manager wants to see it. The production manager can now play WHAT-IF games with his or her capacities, costs, and forecasts and, in addition to allocating capacity, make decisions about acquiring new capacity or eliminating unneeded capacity.

In all of the above cases, EMPIRE, IFPS, and MPSX, it was possible to use SAS as an interface between a non-programmer whose needs did not exactly fit the software package, and a package which was very good at performing specific tasks.

SAS DATA SETS FOR STORAGE

When accumulating data for future use by SAS programs, whether for one-time storage or on a rolling year-to-date basis, the programmer should store the information in SAS data sets. If his or her installation has the capability, these should be in generation data groups. Figure 12 shows how this aids in tracking changes which may have been unforeseen at the time the program was originally written.

By having the data stored in SAS data sets, the programmer only needs to use the INPUT statement for new data and this improves the efficiency of regularly run programs. Furthermore, by using PROC CONTENTS and PROC PRINT, a new programmer can quickly become familiar with the existing files and minimize the time spent "re-inventing the wheel".

If changes are needed in the data, the user can be shown how to use PROC EDITOR, with its find, add, replace, and delete capabilities. Thus, the user can get right into the data to determine which changes are necessary, and make those changes. PROC COPY will allow a disk data set to be placed on tape for long-term storage and will allow the reverse to take place when testing or modification is necessary.

In addition, it is sometimes necessary to strip large company files for small subsets of data which are then input into SAS programs. In these cases, a general retrieval package such as SYNCSORT can be very helpful in cutting down the input-output time and the sorting time which occurs when SAS reads and formats the file.

THE PROGRAMMER-USER INTERFACE

When introducing a new method into a large organization, it is often both necessary and desirable to do it through the use of a pilot program, or prototype. Any errors in the original design can be caught before the method is widely implemented, and the original design can be modified to meet unforeseen needs.

Prototyping has been referred to as "middle-out" design. First, the basic body of the system is worked out. Then, the rest of the system is completed in a dynamic atmosphere.

One of the most important things to do when designing the system is to attempt to find out everything which the user wants from the system and to find out all of the changes which the user is going to want to make. Also, it is important to find out the informal, unwritten structure in cases where the new system is going to replace an old one. If the new system only reproduces the official structure, and a major chunk of the work is actually being done on an informal, ad hoc basis, much time and money will have been wasted.

Using SAS lends this type of flexibility to a project. The input checking, the use of MERGE with the missing value checking, and a simple PROC PRINT will help find all of the holes in the data which are officially not there. PROC MEANS, and in some cases PROC GLM, PROC FREQ, and PROC DATACHK (contained in the supplemental library), can be used to test the data for "reasonableness". As the user becomes familiar with how the system is evolving, he or she can suggest changes and point up unusual concatenations of circumstances which may need to be accounted for. This becomes crucial as the user realizes that many situations which "scarcely occur and can be dealt with as they come up" need to be accounted for by the system, even if only to point out to the manager that the situation has indeed occurred and does need to be dealt with.

The user also has a key role in pointing out to the programmer which situations are local to his or her sections and which are global to the organization. The organization may desire to have certain global parameters (i.e. the beginning and end of the fiscal year) fixed and unchangeable while it may be useful to leave individual managers responsible for certain other parameters, such as job categories. Although these are questions which the programmer always needs to ask, a good user can be helpful in pointing out these situations.

A good user, in fact, is crucial to successful prototyping. The user must be willing to question current methods, willing to change the way he or she operates, willing to devote a large amount of time to developing the system, and committed to the successful outcome of the project. If the user does not become involved in project development, it is possible for a technically elegant system to emerge which does not meet the organization's needs and remains unused.
while the informal structure continues to operate. In certain extreme cases the new system will become a hindrance and the user will need to spend extra time reconciling the traditional, informal system with the new, unused system.

Since it is necessary for other managers to want to use the new system, it will only spread through the organization if the other managers perceive the first user gaining from using the prototype. This will not happen if the first user finds himself or herself bogged down in reconciling the new system to the actual business environment. For the first user to be able to use the new system effectively, he or she must be intimately involved with how it functions, although not necessarily in the programming sense. He or she must know how the system handles both normal and special cases, and which aspects are fixed and which are under the user's control.

The user will probably want to install the system in pieces, so it is advisable to turn over the system in pieces. The development of later aspects may be directly dependent on the knowledge gained earlier. SAS, with its neat divisions into DATA and PROC steps and its easy-to-use input and output features, is eminently suitable for this type of modular development.

In the early stages, merely getting the user the numbers to be typed into a report can save a lot of time. In later stages, report-formating and iterative processing can round out the project. In any case, whenever other managers perceive the prototype user making better use of his or her time as a result of the system, the system's credibility increases throughout the organization and more people will want to use it.

FIGURE 1
a) PROC PRINT;
b) PROC PRINT;
   TITLE LISTING OF AGED A/R FILE;
c) PROC PRINT;
   VAR CUSTOMER UNDER30 30 60 60 90 OVER90;
   TITLE LISTING OF AGED A/R FILE;
d) PROC PRINT;
   ID CUSTOMER;
   VAR UNDER30 30 60 60 90 OVER90;
   TITLE LISTING OF AGED A/R FILE;
e) DATA NULL:
   FILE PRINT NOTITLES HEADER=NEWPAGE;
   PUT @ 1 CUSTOMER $CHAR20.
   @ 30 UNDER 30 DOLLAR10.2
   @ 40 30 60 DOLLAR10.2
   @ 60 60 90 DOLLAR10.2
   @ 75 OVER 90 DOLLAR10.2;
   TOTAL = UNDER 30;
   TOT30 60 + 30 60;
   TOT60 90 + 60 90;
   TOTOV90 + OVER90;
   IF EOF THEN PUT /93 'TOTAL'
   @ 28 TOTAL DOLLAR12.2
   @ 43 TOTUN30 DOLLAR12.2
   @ 56 TOT60 DOLLAR12.2
   @ 73 TOTOV90 DOLLAR12.2;
   RETURN;
   NEWPAGE:
   PUT 0 40 'LISTING OF AGED A/R FILE'/
   0 10 'CUSTOMER'
   0 30 'UNDER 30 DAYS'
   0 40 '30 TO 60 DAYS'
   0 60 '60 TO 90 DAYS'
   0 75 'OVER 90 DAYS'/;

FIGURE 2
DATA MACHINES NO PROD NO PARTS;
MERGE MACH PROD (IN=PROD) MACHPART (IN=PART);
IF PROD AND PART THEN OUTPUT MACHINES;
IF PROD AND PART THEN OUTPUT MACHINES;
IN PROD AND PART THEN OUTPUT NO PROD;
PROC PRINT DATA=NO PROD;
TITLE MACHINES WITHOUT SCHEDULED PRODUCTION;
PROC PRINT DATA=NO PARTS;
TITLE MACHINES WITHOUT ALLOCATED PARTS;
**FIGURE 3**

DATA COSTS (KEEP=MACHINE COST ITEM)
FORECAST (KEEP=ITEM DOLLARS)
REVENUE (KEEP=ITEM PRICE);
INPUT @ 1 CODE @;
IF CODE = 1 THEN DO;
  INPUT 03 MACHINE $CHARIO.
  @ 20 COST 15-18 2
  @ 20 OUTPUT COSTS;
END;
IF CODE = 2 THEN DO;
  INPUT 03 ITEM $CHAR 3.
  DOLLARS 5-8;
  OUTPUT FORECAST;
END;
IF CODE = 3 THEN DO;
  INPUT 03 ITEM $PRICE 5-9 3;
  OUTPUT REVENUE;
END;

**FIGURE 4**

<table>
<thead>
<tr>
<th>DIVISION</th>
<th>DIVISION NUMBER</th>
<th>UNITS IN INVENTORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHIRTS</td>
<td>058</td>
<td>200000</td>
</tr>
</tbody>
</table>

**FIGURE 5**

// JOB accounting-info
// EXEC SAS
// FILE DD DSN=INPUT.FILE,DISP=SHR
// SYSSIN DD*
MACRO HEADER 'JULY, 1979' %
DD DSN=SAS.PROGRAM,DISP=SHR

DATA NULL;
SET DATA;
FILE PRINT HEADER = NEWPAGE;

NEWPAGE: PUT 'BILLINGS AS OF' HEADER;

**FIGURE 6**

// JOB accounting-info
// EXEC SAS
// FILE DD DSN=INPUT.FILE,DISP=SHR
// SYSSIN DD*
MACRO SORTTYPE
  BY AREA CUSTOMER; %
DD SAS.PROGRAM,DISP=SHR

PROC SORT;
SORTTYPE
PROC MEANS N SUM MEAN STD;
SORTTYPE
TITLE CUSTOMER BILLINGS;
TITLE SORTTYPE

**FIGURE 7**

FLOWCHART OF ITERATIVE PROCEDURE
ALLOCATING PERMANENT INVESTMENT TO
DIVISIONS HAVING SALES FOR CALCULATIONS
OR ROI

- Read Cost Accounting System: Obtain
  Depreciation Allocation Percent from
  Manufacturing to Other Divisions
- Calculate Formulas for Allocation of
  Permanent Investment from Each Manu-
  facturing to Other Divisions
- Read Cost Accounting System to Obtain
  Each Division's (Manufacturing or Sales)
  Permanent Investment
- Allocate Permanent Investment from Each
  Manufacturing Division to Other Manu-
  facturing Divisions or Sales Divisions

  Have All
  Iterations
  Been Performed

  (The program section in Figure 8)

- Read Financial Statements for Profits
  of Each Division
- Report ROI for Each Sales (or Manu-
  facturing with Sales Division)
FIGURE 8

// JOB accounting info
// EXEC SAS
// DD DD COST,ACCTG,DISP=SHR
// SYSIN DD *
// MACRO ITERATE
// ALLOCATE
// ALLOCATE
// ALLOCATE
// DD SAS.PROGRAM,DISP=SHR

MACRO ALLOCATE
DATA SENDER;
  SET BEGIN;
  *ALLOCATE THE RECEIVED INVESTMENT;
  GO OUT=TO GO * RATIO * PCTTODIV;
DATA RECEIVER;
  SET SENDER;
  GO IN = GO OUT;
  *TAKE IN THE ALLOCATED INVESTMENT;
  LEDGER = NEWLED;
PROC SORT;
  BY LEDGER;
PROC MEANS SUM NOPRINT;
  BY LEDGER;
  VAR GO_IN;
  OUTPUT OUT=RECEIVER SUM=GO_IN;
PROC SORT DATA=SENDER;
  BY LEDGER;
PROC MEANS SUM NOPRINT;
  BY LEDGER;
  VAR GO_OUT;
  ID HOLDER;
  DATA CALC;
  OUTPUT OUT=SENDER SUM=GO_OUT;
DATA BEGIN;
  MERGE SENDER RECEIVER;
  BY LEDGER;
  HOLDER=SUM (HOLDER,GO_IN,(-1 GO OUT));
  TO_GO=GO_IN;
  DROP GO_IN GO OUT;
DATA BEGIN;
  MERGE BEGIN RATIO;
  *SET THE STAGE FOR THE NEXT ITERATION;
  BY LEDGER;
  ITERATE
  NOTE: The data sets WORK.RATIO and WORK.BEGIN were created by the program before the macro.

FIGURE 10

// JOB accounting info
// EXEC SAS
// ALLOC DD *
// the coded input cards
// SYSIN DD SAS.PROGRAM,DISP=SHR

FIGURE 11

PROC MATRIX;
  FETCH WORKCALC DATA=WORKER COLNAME=P LINE;
  DO I = 0 TO NROW (WORKCALC) - 20 BY 20;
  WORKCALC(I+5,) = WORKCALC(I+4,) +
  WORKCALC(I+8,) = WORKCALC(I+7,);
  WORKCALC(I+18,) = WORKCALC(I+13: I+16,)
  (*);

FIGURE 12

// JOB accounting info
// EXEC SAS
// BACK1 DD SAS.DATASET(D), DISP=SHR
// BACK2 DD SAS.DATASET(-1), DISP=SHR
DATA; SET BACK 1 (IN=IN1) BACK 2 (IN=IN2);
  BY JOBORDER;
  IF FIRST. JOBORDER AND LAST.JOBORDER;
  IF IN1 THEN TYPE = 'NEW ORDER';
  IF IN2 THEN TYPE = 'MISSING';
PROC PRINT;
  VAR JOBOYER TYPE;
  TITLE JOBOYER CATEGORIES WHICH HAVE BEEN;
  TITLE2 DROPPED OR ADDED;

PROC CHART;
  BLOCK VENDOR/GROUP=YEAR
  SUBGROUP=VENDOR
  SUMVAR=PCT;
  TITLE .H=3 F=ITALIC ATHLETIC UNIFORMS;