Clearly, the SAS macro language considerably increased the potential power of the SAS package. Yet macros provided more than just power. At the same time, they promoted the maintainability of SAS-based systems by allowing those systems to be much more structured and modular than before. "Chunking" an application into a set of clearly defined, interrelated macros facilitates later enhancements because any changes are module-specific, and each module can be tested and debugged independently of the system as a whole.

Yet anyone who has struggled with writing and debugging macros of even intermediate sophistication can attest to their problematic nature. One experienced SAS software consultant told me he avoids macros because their maintainability is so difficult. Triple ampersands, quoting functions, and the often confusing production of SAS code all contribute to making a complex macro virtually impossible to read.

This paper attempts to show concretely that the desirable goals of maintainability and modularity can indeed be contradictory with respect to SAS macros. By presenting an extremely complex macro and outlining the reasons for its existence, the paper will make a case for the following unintuitive syllogism:

- Modular construction tends to make computer systems more maintainable.
- Macros tend to make SAS-based systems more modular.
- However, complex macros do not always make SAS-based systems more maintainable.

INTRODUCTION

Pacific Gas and Electric (PGandE) has for several years had a set of SAS programs, dubbed CLASSKW, which extrapolate the electrical usage data of individual customers to the customer group level. Those aggregated data are then used to determine what portion of total system generation is going to each customer group (residential, agricultural, small commercial, etc.). In order to sum the raw data appropriately, it is of course necessary to have an established way of categorizing any given customer into the correct subgroup.

Theoretically, dividing customers into these subgroups should not be difficult. Unfortunately, PGandE has 3.7 million customers on some 30 to 40 different basic rate schedules, and major inconsistencies in categorization have crept in over the years. For a variety of reasons, a given customer might be labeled as small light and power for one purpose and medium light and power for another purpose. Intricate contractual relationships often make a clear identification of the appropriate group in a given case quite difficult. Not only are the customer group determination criteria sometimes inconsistent with one another, they tend as well to change from year to year as rate schedules are added or dropped or as new considerations emerge.

In the old CLASSKW system, the process of placing a customer into the appropriate customer group was performed by in-line code, consisting basically of IF-THEN-ELSE statements. Various subprograms of the CLASSKW system had slightly different versions of this code, owing to the system being modified from year to year or weak to week by non-programmers. Standardization and modularization was badly needed, as well as a clear way to change the categorization criteria easily and without touching the actual code. The GETCLASS macro, with its associated macro MAKEVARS, was created to answer this need. With these two macros, all categorization criteria are actually contained in a special SAS data set (CLSGROUP), which can be browsed or modified by the user with SAS/FSP. The new CLASSKW system converts the CLSGROUP data set to macro variables (via the MAKEVARS macro) and builds, via the GETCLASS macro, SAS code which contains the appropriate categorization logic as defined by the current CLSGROUP data set.

CLSGROUP SCREEN

The two basic criteria used to categorize any customer at PGandE are the rate schedule (ESCHED, with 35 basic values) and the electric revenue account number (EREVACCT, with 10 values). These two variables, obtained from a large company database external to CLASSKW, have a high level of correlation but are not wholly consistent. In addition, other variables might be needed for particular categories in order to take into account anomalies such as special hand-billed customers. An example of the customer grouping screen is shown in Figure 1. CLASSKW used about 25 such CLSGROUP observations were filled out using SAS/FSP.

The screen layout allows the user to define the customer group name and to specify certain relevant attributes of that group which will affect the processing later on, such as whether it is a resale, retail, or a public authority customer group. The actual categorization logic comes from the combinations of rate schedule (ESCHED) and electric revenue account number (EREVACCT) which are filled in by the user, along with a longer SAS code field.
which can be used, for example, for unexpected new criteria which are independent of ESCHED and EREVACCT. Up to five different numbered levels are provided so that the user can specify alternative combinations. The different numbers are alternatives and are thus linked by a logical OR in the final generated code.

SAS code will ultimately be generated matching the criteria provided on these screens by the user; Figure 2 shows the code that is equivalent to the screen shown in Figure 1 so that the reader can have a clearer idea of how the definition screen works. The macro GETCLASS will generate this code, although not in such nicely readable, indented form as I have provided here.

MACROS

Once the SAS/FSP screen has been filled in as outlined above, the CLASSXM system can use, at various points in its calculations, the SAS data set which they represent to generate the appropriate categorization code. Two steps are involved in that generation. First, the SAS data set, CLSGROUP, must be converted to SAS macro variables. Second, the required IF-THEN-ELSE chain must be generated to reflect the categorization criteria provided on the screens by the user. The first of these steps is achieved via the MAKEVARS macro.

MAKEVARS MACRO

Each of the fields on the categorization screen is, of course, a variable in the SAS data set CLSGROUP. MAKEVARS reads those variables for each observation (where each observation represents a different customer group) and converts them to macro variables by using the SYMPUT function.

Since there are about 25 customer groups and thus 25 observations, the SAS variables have the same names for each customer group. However, each macro variable must have a unique name, for all must be available to the macro processor at the same time. To achieve such uniqueness, MAKEVARS uses the first three characters of the customer group name as part of the name of all the macro variables associated with that group. Over 100 macro variables are created for each customer group. Fortunately, we are able to run with a large region size to accommodate this enormous symbol table.

MAKEVARS thus simply sets up the capability for subsequent data steps to use the GETCLASS macro. It forms a separate DATA step, unlike GETCLASS, and needs to be called only once in any job which will require GETCLASS.

For the source code to MAKEVARS, see Figure 3.

GETCLASS MACRO

GETCLASS generates the appropriate and lengthy IF-THEN-ELSE statements within a user-defined DATA step, so that the CLASS variable will be set according to the current customer group definitions taken from the SAS data set CLSGROUP by the macro MAKEVARS. Assuming that the variables ESCHED and EREVACCT (along with any other variables required for categorization) have been given values prior to the calling of the macro, GETCLASS will cause the variable CLASS to be given the value which corresponds to the values of those variables. In short, GETCLASS functions somewhat like an advanced table lookup routine.

GETCLASS generates an IF-THEN statement for each customer group. Within each customer group (see Figure 1), it needs to (conceivably) create code for up to three different portions for each of up to five different levels. These three portions consist of:

- the CODE portion, where a SAS statement may have been entered on the customer group definition screen;
- an ESCHED portion, which will handle the various forms of the ESCHED variables allowed for on this level;
- an EREVACCT portion, which will handle the various forms of the EREVACCT variable allowed for on this level.

Appropriate ANDs and ORs, as well as correctly matched parentheses, must also be generated, depending on which portions have been selected for a given level.

GETCLASS does allow a "catch-all" for unanticipated combinations of parameters. Should such a combination cause the IF-THEN chain to fall through, one last ELSE group causes the variable CLASS to simply receive the value of EREVACCT, and the variable ANOMALY is flagged as a warning.

Figure 4 shows the source code to the GETCLASS macro; Figure 5 shows a section of the code, as produced in the SAS log, which GETCLASS actually generated from the 1985 customer group definitions. Note that the actual machinations of customer group determination are transparent to the calling program; the main thing is that the variable CLASS has received the correct value.

CONCLUSION

A quick perusal of Figure 4, the source code to GETCLASS, will underscore the points made in the abstract of this paper concerning the maintainability of macros.
GETCLASS has without a doubt made the new CLASSKW system considerably more able to accommodate the inevitable changes in customer group definitions, and made it possible for non-programmers to alter those definitions with as few side-effects as possible. Compared to the scattered and inconsistent code available before, GETCLASS is more modular, more expandable, and more provably correct.

However, the question remains as to whether the assets outweigh the liabilities. GETCLASS is so difficult to read and follow, so dependent on macro language "quirks," that even the most experienced SAS programmers in our group will find it arduous to maintain or revise.

I have no solution to this dilemma; many arguments speak for the merit of utility over readability in certain cases, and if GETCLASS has been designed with the kind of flexibility I intended, revision should not be a major issue. Yet I still wonder if the supreme power of the SAS macro facility may lead other users into the same Catch-22, where disaster is just another ampersand away.

For additional information, contact
Peter Kretzman
Rate Department
Pacific Gas and Electric Company
San Francisco, CA 94106
(415) 972-6449

ACKNOWLEDGMENTS: Other members of the CLASSKW project team contributing to its overall design were Theresa L. Kervin, D. Scott Litsy, N. C. Neustrup, and Ann K. Smith.

NOTE: SAS and SAS/FSP are registered trademarks of SAS Institute, Inc., Cary, NC, USA.

FIGURE 1: Sample screen defining customer group AGR

<table>
<thead>
<tr>
<th>CLASSKW CUSTOMER GROUPING DESIGN SCREEN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer Group: AGR</td>
</tr>
<tr>
<td>Multimetered: Y</td>
</tr>
<tr>
<td>Retail: Y</td>
</tr>
<tr>
<td>Resale: N</td>
</tr>
<tr>
<td>Pub. Auth.: N</td>
</tr>
<tr>
<td>CLASSKW processing level: C</td>
</tr>
<tr>
<td>Default Voltage Level: SECONDARY</td>
</tr>
<tr>
<td>ESCHED values</td>
</tr>
<tr>
<td>ERREVACCT values</td>
</tr>
<tr>
<td>1) PA 1 PA 2 PA R A 354 360 N D</td>
</tr>
<tr>
<td>2) A N</td>
</tr>
<tr>
<td>3) A N</td>
</tr>
</tbody>
</table>

FIGURE 2: Code generated by GETCLASS from screen in Figure 1.

IF (ESCHED = "PA 1" OR ESCHED = "PA 2" OR ESCHED = "PA R") AND (ERREVACCT = 354 OR ERREVACCT = 360) THEN
CLASS = "AGR";
**FIGURE 3:**

MAKEVARS macro code

xmacro makevars;
options missing='.';

data _null_; set dbn.clsgroup end=eof;
array erev(i) erev1_1-erev1_10 erev2_1-erev2_10 erev3_1-erev3_10 erev4_1-erev4_10 erev5_1-erev5_10;
array esch(i) esch1_1-esch1_10 esch2_1-esch2_10 esch3_1-esch3_10 esch4_1-esch4_10 esch5_1-esch5_10;
array code(lev) code1-code5;
if eof then do;
last = _n_ + 1;
lifetime = 'GROUP' || left(trim(last));
call symput(lastname,' ');
put 'VARS complete ' lifetime=' last= grouping= lastname=';
end;

万吨 to 5;
cname = trim(substr(grouping,1,3)) || 'COD' || left(trim(lev));
call symput(cname,code);
end;
do over erev;
lev = int((i-1)/10) + 1; /* focuses on the 2 in erev2_1;*/
val = i - (lev-1) X 10; /* focuses on the 1 in erev2_1;*/

万吨 to 5;

万吨 to 5;

call symput(rname,left(trim lucrav)));
call symput(sname,esch);
if val=10 then do;
万吨 a dummy at end;

万吨 to 5;

call symput(rname,' ');
call symput(sname,' ');
end;
run;

xend makevars;

516
FIGURE 4:
GETCLASS macro code

%macro getclass;

IF

let g=1;
Xdo Xwhile (&&group&g ne _); %* looks at ampers group1 etc.;
let gp=Xsubstr(&&group&g,1,3);
Xif Xsubstr(&gp,3,1)= Xthen
  Xlet l=1;
  Xlet v=1;
Xdo Xwhile (((length(&&&gp.S&l._&v) ne 0 )
or (&&&gp.R&l._&v ne _ )
or (length(&&&gp.CODE&l) ne 0 )); %* main loop;
let schflag=0;

%* SAS CODE portion;

Xif length(&&&gp.CODE&l) ne 0 Xthen
Xdo;
  (&&&gp.CODE&l)
Xif (length(&&&gp.S&l._&v) ne 0 
or &8&&&gp.R&l._&v ne _ ) Xthen
Xdo;
  AND
  Xend;
Xend;

%* ESHC portion;

Xif length(&&&gp.S&l._&v) ne 0 Xthen
Xdo;
  Xlet outvar=qquote(x(ESHCED='&&&gp.S&l._&v'));
  Xunquote(&outvar)
  Xlet schflag=1;
  Xend;
Xlet v=2;
Xdo Xwhile (length(&&&gp.S&l._&v) ne 0 );
Xdo;
  Xlet outvar=qquote(OR ESCHED='&&&gp.S&l._&v');
  Xunquote(&outvar)
  Xend;
  Xlet v=xeval(&v + 1);
Xend;
Xif &schflag=1 Xthen
  Xdo;
  )
Xif &&&gp.R&l._&v ne _ Xthen
Xdo;
  AND
  Xend;
  Xend;

%* EREV portion;

Xlet revflag=0;
Xif &&&gp.R&l._&v ne _ Xthen
Xdo;
  (EREVACCT='&&&gp.R&l._&v
Xlet revflag=1;
Xend;
Xlet v=2;
Xdo Xwhile (&&&gp.R&l._&v ne _ );
Xdo;
  OR EREVACCT = &&&gp.R&l._&v
Xend;
  Xlet v=xeval(&v + 1);

Xend;

517
FIGURE 5
Actual generated code from GETCLASS macro

LENGTH CLASS $S1
IF (ESCHED='PA') AND (EREVACCT=354 OR EREVACCT=360) THEN CLASS = "AGRIC"
ELSE IF (GROUP='SLAC') THEN CLASS = "AMESLAC"
ELSE IF (ESCHED='A') AND (EREVACCT=352 OR EREVACCT=360) OR (ESCHED='A0') AND (EREVACCT=352 OR EREVACCT=360) OR (ESCHED='A 19') AND (EREVACCT=352 OR EREVACCT=360) THEN CLASS = "A1"
ELSE IF (ESCHED='A 12') AND (EREVACCT=355 OR EREVACCT=360) OR (ESCHED= 'AD') AND (EREVACCT=355 OR EREVACCT=360) OR (ESCHED='A 20') AND (EREVACCT=355 OR EREVACCT=360) THEN CLASS = "A2"
ELSE IF (ESCHED='A 18') AND (EREVACCT=359 OR EREVACCT=360) THEN CLASS = "A18"
ELSE IF (ESCHED='A 21') AND (PVD <= 1) AND (EREVACCT=353 OR EREVACCT=360) THEN CLASS = "A21"
ELSE IF (ESCHED='A 22') AND (EREVACCT=353 OR (ESCHED='A 23') AND (EREVACCT=359) THEN CLASS = "A22"
ELSE IF (EREVACCT=359) THEN CLASS = "BART"
ELSE IF (ESCHED='A 21') AND (3 <= PVD <= 4) AND (EREVACCT=353 OR EREVACCT=360) THEN CLASS = "B21"
ELSE IF (GROUP='CPM') THEN CLASS = "CPM"
ELSE IF (GROUP='CVP') THEN CLASS = "CVP"
ELSE IF (ESCHED='A 21') AND (PVD=3) AND (EREVACCT=353 OR EREVACCT=360) THEN CLASS = "D21"
ELSE IF (EREVACCT='A 21') AND (PVD <= 1) AND (EREVACCT=359 OR EREVACCT=360) THEN CLASS = "D21"
ELSE IF (EREVACCT='A 21') AND (3 <= PVD <= 4) AND (EREVACCT=359 OR EREVACCT=360) THEN CLASS = "E21"
ELSE IF (EREVACCT='A 21') AND (PVD=5) AND (EREVACCT=359 OR EREVACCT=360) THEN CLASS = "F21"
ELSE DO;
CLASS=LEFT(EREVACCT);ANOMALY=1;LENGTH ANOMALY 2;END;
END;

X* see if there is another level;
Xif ((Xlength(&&&gp.581._l) ne 0 )
or (&&&gp.R81._l ne )
or (Xlength(&&&gp.COD81) ne 0)) Xthen
Xdo;
Xend;

X* get set for next grouping;
Xif &group&g ne Xthen
Xdo;
ELSE IF
Xend;

X* end of emper group do block:
Xdo;

X* if case not accounted for;
ELSE DO,CLASS=LEFT(EREVACCT);ANOMALY=1;LENGTH ANOMALY Z;END;

X* end of main do loop;
Xdo;

Xmend getclass;

FIGURE 5:
Actual generated code from GETCLASS macro