SAS® Macro Solutions: An Application to Get You Going  
Ralph W. Leighton, The Hartford, Hartford CT

1.0 Whither We Go ...?

In one sense this is the story of an application -- a self-contained one which happens to heavily use basic features of the SAS Macro language. In another sense, this is a tutorial on how basic features of the SAS Macro language can be combined to solve a non-trivial problem in a relatively clean manner.

The application is a “flexible loss development triangle facility”, which serves to make reports and files for spreadsheet users; these individuals work in an actuarial area of a property-casualty insurance company. The key elements of the macro language used in this application are:

- Macro variables embedded in SAS Code, to flexibly dimension indexed lists and arrays;
- Macro variables as global parameters;
- Setting macro variables in DATA steps using CALL SYMPUT;
- The macro assignment statement, %LET;
- Macro functions: e.g. %EVAL to perform arithmetic, and %SCAN to parse a character string;
- The use of %IF ... %THEN %DO....groups;
- The use of iterative %DO loops; and
- Within the last, how one creates and uses the analogue to arrays in the SAS macro language.

The present exercise also has some secondary features of possible interest:

- The triangle application is run from a very simple SAS/AF® PROGRAM entry.
- The application makes a key use of “meta-data” (Data about data) to drive computations.

But first, as background, we provide a brief insurance lesson explaining the raison d'être of the triangle facility.

2.0 Loss Development Triangles

Property / Casualty insurance provides coverages to individuals for personal automobile and homeowners’, and it provides coverages to commercial customers for property, liability, commercial auto (i.e. commercial vehicle) and workers’ compensation. An insurance company is very unlike a manufacturing operation or many services. In those, the costs of operation are incurred up front or simultaneous with the selling of the product or the distribution of the service. In insurance, the "product" sold is the insurance policy, and the largest cost borne by the seller arises from the claims that come in, after the sale of the policy -- sometimes a very long time afterwards.

Thus a key problem in the monitoring of the profitability of the product sold -- and the level of loss reserve liabilities to be carried on the balance sheet -- is the continual need to estimate the ultimate costs against the policies sold. This analysis -- usually performed by or guided by actuaries -- often starts by building information as to how these costs have emerged in the past. A common way of showing these patterns of emerging costs is the loss development triangle.

A simplified example of one of these triangles appears in a figure at the top of the next page. The horizontal axis is the “accident year”, the year in which the incidents occurred that spawned the claims. The vertical axis represents regular valuation points, at 6-month intervals, of the total payments made to claimants to date on the particular accident year. Eventually, this series of valuations approaches an ultimate value -- an asymptote. This is the final cost of the accident year, and it is the actuary's objective to determine the ultimate costs of the recent, more immature, accident years.

In a simplified two-dimensional form, the data in these triangles could be viewed as a sequence of records (observations) having the following logical keys:

- **LINE** - Line of Business Ident Code
- **ITEMNAME** - Loss Variate Ident Code
- **ACCYEAR** - Accident Year of Loss

The data on each observation consists of a series of calendar valuations of the "item" referenced in the key ITEMNAME:

- LSS1, LSS2, LSS3, ...etc. ...

In the application soon to be described, these are cumulative evaluations at **monthly** intervals,
starting with the first month of each accident year and stretching out to the current reporting month. The "item" is a quantity such as: total payments to claimants; numbers of known claims; totals of settled cases to date; and total losses reported (incurred) whether or not as yet paid out. This is standard property-casualty insurance data, and data like it, at annual development points, is publicly reported in such documents as the N.A.I.C. Annual Statement ("Yellow Book") and the 10-K filings for the S.E.C.

3.0 Nature of the Application

The application to be described platforms triangle displays of insurance loss data items, like the paid loss display at the top of the page. Using a basic set of 25 data quantities -- loss "variates" -- one can create triangles of some 125 or so derived items with an insurance meaning.

Four examples of the basic starting data items are:

- Incurred Losses -- reported amounts and changes in same, as filed by claimants.
- Paid Indemnity Losses -- amounts paid out to claimants from the losses incurred. (That's what the display at the top of the page shows.)
- Paid Allocated Expenses -- amounts paid out to claim adjusters, lawyers, physicians, and other individuals hired to assist in addressing claims.
- Reported Claims -- the number of claims the company knows about to date.

From these four, other quantities can be formed, for example:

- Average Claim Cost -- the ratio of the incurred value to the reported claims.
- Expense Ratio -- the ratio of the paid allocated expenses to the payments to the claimants.
- Unpaid Losses -- the amount of Incurred Losses not yet paid.

The loss triangle application needs to satisfy the following modest user requirements:

Give the user the option to request a report or a spreadsheet importable '.dat' file.

Enable the user to generate the reports and/or '.dat' files, from an easy-to-use screen.

Give the user total freedom to select the lines (i.e. the subsets of the company's business) he wants.

Give the user the total freedom to select the items on the report (or file). These are the "variates", like incurred losses or paid allocated expenses.

The application has a couple of secondary objectives that have nothing directly to do with the user's requirements, but do bear upon its future maintenance:

To table drive the process as much as possible -- i.e. to leave the processing code as immune to change as is feasible.

To set up the application in as non procedural a fashion as possible.

4.0 Launching the Application using SAS/AF®

The application is operated interactively by a simple, low technology SAS/AF PROGRAM entry in the DEC Alpha Mid-tier environment. This entry, whose screen is shown at the top of the next page, captures the following parameters:

- LINELIST - Sub-lines desired for analysis
- ITEMLIST - The variates desired
- INFILE - Source SAS Dataset to be used
- OUTFILE - Output "dat" file if any
- LASTDEVL - Highest development point on reports
- ALLYEARS - Show all years or only recent 13
- BYMONTH - Developments by month or quarter

The last two are "yes-no" switches handled as SCL Choice groups on the SAS/AF screen:

Group YRCHOICE --> RECENT, ALLYEARS.
Group DVCHOICE --> BYQTR, Bymonth

Behind the screen above, the SCL code below launches the application as a macro invocation, the macro's name being YR0141PC:

LENGTH BYMO ALL $3 ;
INIT: YRCHOICE=RECENT;
DVCHOICE=BYQTR;
RETURN;
MAIN: RETURN;
TERM: IF _STATUS_ = "C" THEN RETURN;
IF YRCHOICE = RECENT
THEN ALL='NO';
ELSE ALL='YES';
IF DVCHOICE = Bymonth
THEN BYMO='YES';
ELSE BYMO='NO';
SUBMIT CONTINUE;
%YR0141PC(LINELIST=&linelist,
ITEMLIST=&itemlist,
INFILE=&fileid,
OUTFILE=&outfile,
ALLYRS=&all,
LASTDEVL=&mx,
BYMONTH=&bymo) ;
ENDSUBMIT;
RETURN;

The remainder of the paper will show how the macro works.

5.0 The Flow of the Macro:

A serious, but common, problem confronts the application: the data received or extracted is neither in the form most desirable for manipulation (i.e. computation) nor in the form the user wants it. And neither of these two latter forms are the same. Specifically:

- As noted earlier, the variates on the source data are in data vectors LSS1, LSS2, .... etc wherein the variate or item is designated by the record as a logical key, ITEMNAME.
- But many, if not most, of the variates are derived computations, and computations are most easily set up using SAS variables -- i.e. with the items or variates as columns, not rows.
- In the final reports (see previous page), the user wants to reverse the role of the development month and accident year from that on the input file: in the input data, accident years are rows whereas on the output they are columns.

So, the following sequence of steps would appear to be in order:

- Extract the data for the LINELIST which the user requested. As noted above, keys are item name and accident year. The record attributes are variables constituting an indexed list of developments by development month. (Section 8.0)
- Transpose (rotate) the data so that the items become SAS variables and the development months become record instances. (Section 8.0)
- Perform the computations -- create only those items requested by the user and place on a (temporary) output file. The macro variables created in the first step are used here to generate a "rubber" DATA step. (Section 9.0)
- Again transpose the data, this time so that the items (variates) are again record identifiers. The record attribute SAS variables become an indexed list of accident year contributions for the development month identifying the record (observation). (Section 10.0)
- Set up the data for the reports and print the reports. (Section 11.0)
- Optionally create the output '.dat' data set for the user to upload into a spreadsheet. (Section 12.0)

Embedded in the third step is the conversion of the user's ITEMLIST request into Indexed Lists of Macro Variables.

6.0 Necessary Globals

To keep an application such as this flexible and immune to change, we need to address the problem of the data vectors: those used by the application and those ultimately produced on the way to final reports and output files. The input loss data has data vectors of the form:

LSS1, LSS2, LSS3, .......

indexed by development month. The output data will also be an indexed list: each observation or record will consist of valuations, as of a fixed development month, of a range of accident years, from some starting accident year (e.g. 1976) as per:

AYR1976, AYR1977, AYR1978, .......

If the first accident year in the data is 1976, then, as of October 1996, the maximum extent of the of the LSSnn entries is 262, the number of months of development on accident year 1976. By December this maximum extent has increased to 264. Thus the upper bound of the LSSnn entries changes with the production date of the source data.

Since the indexed lists are going to be pervasive and since we clearly don't want to buy into code maintenance each time the production month changes, we would benefit from making the list extents variable, and this we can do using three macro variables:

&mospan - the current maximum number of months of development
&firstyr - the first accident year in the data.
&proyear - the current production year (which is also the most recent accident year).

which will mean that our input data vectors will have the form:

LSS1-LSS&mospan.

and the output data vectors will have the form

AYR&firstyr.-AYR&proyear.

O.K. - how do we give values to these macro variables? We could add them as parameters to the
macro, but this seems a little wasteful, since the values they carry really depend on the nature of the input data. So we follow a different course of action: we set up a little system file DATECARD with a single observation containing the first year on the data and the current production date. The latter can be automatically updated as the loss data is refreshed.

Now consider the little DATA-step below:

```
DATA _NULL_;
SET RESRVING.DATECARD;
CALL SYMPUT('PRODATE',PRODATE);
CALL SYMPUT('FIRSTYR',FIRSTYR);
PROYR = FLOOR(PRODATE/100);
CALL SYMPUT('PROYEAR', PROYR);
NOYRS = PROYR - FIRSTYR + 1;
CALL SYMPUT('YRS', NOYRS);
PROMON = MOD(PRODMO,100);
CALL SYMPUT('PROMON', PROMON);
MOS = (NOYRS - 1)*12 + PROMON;
CALL SYMPUT('MOSPAN', MOS);
STOP;
RUN;
```

This can serve as a "start-up" routine, includable as part of the user's AUTOEXEC. Prior to getting into the application, this program executes: it accesses DATECARD and uses the CALL SYMPUT routine to create global macro variables for the application:

7.0 Getting into the Macro - Editing the Parameters

SAS macros begin with a MACRO statement listing the various key word parameters, and this application begins with the following one:

```
%macro SB0605PC(infile=STATMSTR,
    outfile=,
    itemlist=01 04,
    linelist=,
    allyrs=NO,
    lastdevl=132,
    bymonth=NO);
```

Some parameters have default values -- others do not. In the previously displayed SCL code which launches the application, all parameters happen to be referenced. But a different invocation like:

```
%macro SB0605PC(outfile=MYFILE,
    linelist=0101 0302,
    bymonth=YES);
```

will automatically pick up variates '01' and '04', will not produce report data for all years, will use input file STATMSTR, and will present data by month.

The parameters require a bit of up-front editing, and this is carried out in the macro code below, which follows the %MACRO statement:

```
%if &lastdevl gt &mospan.
%then %let lastdevl=&mospan.;
%let sublistlist=;
%if &linelist ne %then %do;
%let first=%scan(&linelist,1);
%if %length(%first) gt 2
%then %let sublistlist=1
%else %let sublistlist=0;
%end;
```

8.0 First Steps : Capturing the Data

The first steps are to extract the data and then "rotate" it for computations (next section). The extract step is fairly simple:

```
DATA EXTRACT (DROP=ARGUM);
SET RESRVSRC.&infile.;
ARGUM = LINE*100 + SUBL;
%if &linelist ne %then %do;
%if &subllist %then %do;
IF ARGUM IN ( &arglist. )
THEN
%end;
%else %do;
IF LINE IN ( &linelist. )
THEN
%end;
OUTPUT;
RETURN;
RUN;
```
"launcher" provide default values for this source. The LINELIST business is more complex.

If LINELIST is not empty and if it consists of four-digit line-subline combinations, then the DATA-step uses one extract criteria, e.g.

\[
\text{IF ARGUM IN (1903 2103 2203)} \\
\text{THEN OUTPUT;}
\]

On the other hand, if two-digit lines are the input, then another form of criteria is used:

\[
\text{IF LINE IN (19 20 21 22)} \\
\text{THEN OUTPUT;}
\]

If LINELIST is empty, all observations on the file are taken. The above is a mini "rubber program" -- what the SAS code looks like (and you can see it on the SAS log if you use the MPRINT option) depends on the parameter string fed to the macro.

The next step sorts the data and rotates it for the computations (next section).

\[
\text{PROC SORT DATA=EXTRACT NODUPKEY; } \\
\text{BY LINE SUBL ACCYEAR ITEMNAME;} \\
\text{RUN;}
\]

\[
\text{PROC TRANSPOSE DATA=EXTRACT OUT=ROTATED NAME=DEVMOCH;} \\
\text{BY LINE SUBL ACCYEAR; } \\
\text{ID ITEMNAME;} \\
\text{VAR LSS1-LSS&mospan.;} \\
\text{RUN;}
\]

PROC TRANSPOSE is an important SAS data manipulation facility that circumvents complex DATA-step code in the tradition of third generation "array" languages like FORTRAN and PL/1. PROC TRANSPOSE changes the significance of the variables (columns) and the rows (record ids), within whatever sort-break criteria (other record ids) the user wants.

In the present instance, the developments are to become record instances and the items or "variates" -- e.g. Paid Losses -- will become the SAS variables. The significance of the parameter usage is:

\[
\begin{align*}
\text{The NAME parameter} & \text{ sets up a SAS variable on the output file DEVMOCH which will capture the name of the transposed column variables. It will have values "LSS1", "LSS2", etc.} \\
\text{The ID statement} & \text{ gives the name of a variable on the input data set containing the SAS variable names to be used on the output "rotated" data sets. ITEMNAME has values like "PAIDIND" (Paid indemnity loss), "CLMCNNTS" (Reported Claim Counts), ... etc.} \\
\text{The BY statement} & \text{ is the sort-break criteria -- the transpositions will be performed within the BY-groups, and all BY variables will be on the output.}
\end{align*}
\]

\[
\begin{align*}
\text{The VAR list} & \text{ has the list of SAS variables on the input dataset which are to be transposed. Note the use of the variable extent &mospan.}
\end{align*}
\]

We are now ready for the computational step.

**9.0 How the "ITEMLIST" is Handled**

It is the intention of the application that the computations be performed on the variables of SAS dataset ROTATED created in the previous step. The actual items requested by the user are in the string ITEMLIST. This can be broken up into individual item numbers -- e.g. '21' for average claim cost -- using an application of the SCAN function.

One way of handling this might be to build a DATA-step in the macro with all possible variate calculations. A SELECT group would be used to match each variate request to the desired calculation. Then the variate results would be placed on an output file CALCFILE, one variate at a time, using the item number ITEMNUMB as a record tag. For example:

\[
\text{DATA CALCFILE} \\
\text{(KEEP = ITEMNUMB ...... RESULT);} \\
\text{RETAINT ITEMLIST ";itemlist.";} \\
\text{LENGTH ITEMNUMB 8;} \\
\text{SET ROTATED;} \\
\text{ITEMNUMB = SCAN(ITEMLIST,1);} \\
\text{DO NN = 1 TO 200} \\
\text{WHILE(ITEMNUM NE .);} \\
\text{SELECT( ITEMNUMB );} \\
\text{:::::::} \\
\text{WHEN(21) RESULT=INCDIND/CLMCNNTS;} \\
\text{:::::::} \\
\text{OTHERWISE;} \\
\text{END;} \\
\text{OUTPUT;} \\
\text{ITEMNUMB = SCAN(ITEMLIST, (NN+1));} \\
\text{END;} \\
\text{SET ROTATED;} \\
\text{RUN;}
\]

Unfortunately, such a DATA-step will require code changes as more variates get defined, either as users get creative or in response to additional data on the input file (parameter INFILE).

An alternative used in the triangle program begins with a table -- a mini example of "meta-data". This table lists each "variate" of interest, both those provided as part of the source data -- e.g. incurred losses -- and those others derivable as a result of arithmetic computation -- e.g. average claim cost.
Ultimately we will need DATA-step code containing arithmetic assignment statements, such as the following statement which computes the ex-catastrophe (i.e. not associated with major weather events) average claim cost (or "severity") as the quotient of the incurred losses divided by reported claim counts:

\[
\text{SEVERITY} = \frac{\text{INCDIND}}{\text{CLMCNTS}};
\]

A "rubber" DATA-step -- as opposed to the alternative in which each variate's calculation is laid out in the SAS code -- will create code for the variables asked for, and nothing else. It will use, indirectly, the metadata table laid out at the bottom of this page.

To this end, we can represent (as per the table) each variate computation in terms of three components:
- the formula for the numerator,
- the formula for the denominator, and
- the scaling factor.

The last is needed for average value computations, since the money amounts on the source file are not in whole dollars but in thousands of dollars. For variates other than averages, the "scale" equals 1. Likewise, for variates which are not ratios, the denominator also equals 1. All source variates on dataset ROTATED -- e.g. the ex-catastrophe losses paid PAIDIND -- are also on the table, since the user may want one or more of these. For them, the numerator merely repeats the variate's name, e.g. PAIDIND, and the denominator and scale are both 1.

VARNUMB is a logical key for the table. For each of the other items in the table, the items have been translated into SAS Formats on a permanent format LIBRARY using PROC FORMAT with the CNTLIN option. VARNUMB is the argument in each case. (See section 12.0 for SAS code illustrating this type of use of PROC FORMAT).

Assuming we have created table VARTABLE and have set up the user defined SAS Formats, the following DATA-step in the application then uses these formats to translate the user's request into indexed sequences of macro variables, one each for the considerations on the table:

```sas
DATA _NULL_;
LENGTH ITEMLIST $150 INDEX $2;
LENGTH NUMER $50 DENOM $50;
LENGTH FORMAT $20 SCALE $8;
LENGTH ITEM 8;
ITEMLIST = "&itemlist.";
JJ = 1;
ITEM = SCAN(ITEMLIST,JJ);
DO UNTIL(ITEM = .);
  VARNAME = PUT(ITEM,VNAME.);
  VARLABEL = PUT(ITEM,VLABEL.);
  FORMAT = PUT(ITEM,VFORMAT.);
  SCALE = LEFT(PUT(ITEM,SCALE.));
  NUMER = PUT(ITEM,NUMER.);
  DENOM = PUT(ITEM,DENOM.);
  INDEX = PUT(JJ,2.);
  CALL SYMPUT('vname'||COMPRESS(INDEX), TRIM(VARNAME));
  CALL SYMPUT('vlabel'||COMPRESS(INDEX), TRIM(VARLABEL));
  CALL SYMPUT('numer'||COMPRESS(INDEX), TRIM(NUMER));
  CALL SYMPUT('denom'||COMPRESS(INDEX), TRIM(DENOM));
  CALL SYMPUT('format'||COMPRESS(INDEX), TRIM(FORMAT));
  CALL SYMPUT('scale'||COMPRESS(INDEX), TRIM(SCALE));
  JJ = JJ + 1;
  ITEM = SCAN(ITEMLIST,JJ);
END;
JJ = JJ - 1;
CALL SYMPUT('novars',
  COMPRESS(PUT(JJ,2.)));
STOP;
RUN;
```

For example, the list of variable names underlying the request then will be in macro variables &vname1, &vname2, etc ... and the list of output formats (to be used later on the users' reports) are in &format1, &format2, etc... The macro variable &novar contains the number of elements in each of the lists, and will be used to control the iteration of %DO loops in the "rubber" DATA step that uses these variables to perform the calculations and produce CALCFILE.

This is the "rubber" DATA step:

```sas
DATA CALCFILE
  (KEEP = LINE SUBL ACCYEAR JDMO LINENAME
   %do jj = 1 %to &novars.;
     &&&vname&jj
   %end; );
LENGTH JDMO 8;
ATTRIB JDMO FORMAT=3.
LABEL="Dev1 Mon";
ATTRIB LINENAME FORMAT=SUBLNAME.
LABEL="..... Description .....";
%do jj = 1 %to &novars.;
ATTRIB &&&vname&jj
  FORMAT=&&format&jj
  LABEL= "&&&vlabel&jj." ;
%end;
SET ROTATED;
LINENAME = LINE*100 + SUBL;
JDMO = SUBSTR(DEVMOCH,4,3);
%do jj = 1 %to &novars.;

```
IF (&&&denom&jj.) NE 0 THEN 
    &&&vname&jj.=(&&&numer&jj.)*
    &&&scale&jj./(&&&denom&jj.);
ELSE &&&vname&jj = 0;
%end;
%if &bymonth NE YES %then %do;
IF   MOD(JDMO,3) = 0  THEN
%end; OUTPUT;
RETURN;
RUN;

First some quick notes as to what is happening here:

- The JDMO variable replaces the character key DEVMOCH which has names like "LSS23" with a numeric variable containing the development month, e.g. '23'.
- The variable LINENAME is a four-position numeric code which has as a format the character description of the line-subline combination.
- The format SUBLNAME referenced in the ATTRIBUTE statement is created from a line of business table using PROC FORMAT with the CNTLIN option, in a manner similar to the VARTABLE based formats.
- The calculations are set up near the bottom of the step, using generic references to the selected variable and its corresponding numerator, denominator and scaling factor.

The "triple ampersand" notation usually raises eyebrows of those encountering it for the first time. The notation is the way one can reference an indexed list of macro variables in the SAS Macro Language in a manner analogous to array references in a SAS DATA-step or in SCL code. The abundance of ampersands arises from the manner SAS uses to resolve a macro value. Consider the first %DO loop listing the variables to be KEEPed on CALCFILE. Suppose we consider the iteration when &jj (the loop counter) equals 3. In the first pass SAS resolves the variable &&&vname&jj to the value &&vname3. On the second pass SAS replaces &vname3 with whatever the DATA _NULL_ step assigned to it using CALL SYMPUT.

If the user supplied only a single item on his/her request list and if it was the item '29' shown by way of example on the table display, then option MPRINT would reveal the following generated SAS DATA-step code on the SAS log:

```
DATA CALCFILE
    (KEEP = LINE SUBL ACCYEAR JDMO LINENAME TOTSEVER);
LENGTH JDMO 8;
ATTRIB JDMO FORMAT=3.
    LABEL="Devil Mon";
ATTRIB TOTSEVER FORMAT=COMMA9.
```

As noted before, this DATA-step component of the macro will not be subject to maintenance as data considerations change. Instead, such maintenance will be shifted to the "meta-data" table, VARTABLE, accessible through FSVIEW or FSEDIT in the SAS/FSP® product.

### 10.0 Preparing the Data for Reports

We now perform the second transposition or rotation of the data. In CALCFILE, the SAS variables are the actual items (variates). What the user wants is report pages (or files) showing for a single item the accident year contributions across the page horizontally by development month. The following code performs the rotation:

```
PROC SORT DATA=CALCFILE;
    BY LINE SUBL LINENAME JDMO;
RUN;

PROC TRANSPOSE DATA=CALCFILE
    OUT=TRIANGLE
    PREFIX=AYR
    NAME=ITEMNAME;
    BY      LINE SUBL LINENAME JDMO;
    ID      ACCYEAR;
    IDLABEL ACCYEAR;
    VAR      %do jj = 1 %to &novars;
        &&&vname&jj
    %end;
RUN;

PROC SORT DATA=TRIANGLE;
    BY      LINE SUBL ITEMNAME JDMO;
RUN;
```

The first PROC SORT rearranges the data since we want to perform the rotation by development month as well as by line-subline. LINENAME is not needed to organise the data, but we need it on the output file and using it as a BY-variable accomplishes that objective.

The rotation (the PROC TRANSPOSE step) specifies that exactly the created variables on CALCFILE be rotated, and the %DO-loop embedded in the VAR statement creates the VAR list. Variable ITEMNAME, referenced in the NAME option picks up the SAS variable name of the variables on the dataset CALCFILE being transposed.
The variables on the output dataset TRIANGLE will be indexed by accident year, and this is taken care of by the ID statement reference to field ACCYEAR on the input file CALCFILE. The ID reference can be sufficient by itself to set up the variable names, if the ID values started with an alpha character. The ACCYEAR values, however, are numeric, and SAS does not allow variables to begin with a numeric -- e.g. '1976' is not a valid SAS variable name. So we use the PREFIX option to plaster "AYR" in front of the accident year. Assuming that data starts with 1976, we will get output variables AYR1976, AYR1977, etc. on the output file TRIANGLE. The IDLABEL statement specifies that the value of ACCYEAR will also provide the SAS Variable Label value, a neat piece of work saving.

Since the data items on CALCFILE may not have the same SAS format, it is likely that the formats of the variables $AYRyyyy$ will be "general" -- i.e. the "logical" formats are now varying by record, not by column (SAS variable). Recovering the format will have to await usage of the data in generating the reports.

11.0 Making the Reports

We are going to make reports that look like the one on the second page of this paper, except that developments will be by month or quarter, not half-year.

The reports are subject to some degree of user control, hence the further instance of "rubber programming" below. The user can terminate developments after a certain level and can suppress the display of older accident years. Because of printer and environment limitations (non-WINDOWS), only 13 accident years will fit on a report gracefully.

To make the reports, we take the "cheap" way out -- the reports will be made by PROC PRINT, after a little data massaging in a DATA step:

```
DATA PRNTFILE;
  ARRAY ADATA(&firstyr.:&proyear.)
    AYR&firstyr.-AYR&proyear.;
  SET TRIANGLE;
  IF JDMO GT &lastdevl THEN DELETE;
  OUTPUT;
  IF MOD(JDMO,12) = 0 THEN DO;
    JDMO = .;
    DO JJ = LBOUND(DATA) TO HBOUND(DATA);
      ADATA(JJ) = .;
    END;
  OUTPUT;
  END;
RETURN;
RUN;
```

This preliminary DATA-step serves two functions:

- It censors the data beyond the highest level of development desired by the user; and
- (A "stupid SAS trick") It introduces a bit of formatting in the data as printed by PROC PRINT. We want to introduce a line of "white space" after each year's worth of developments, and outputting an extra line of missing values serves the same purpose.

As noted before, the variates have different formats. Dollar amounts are comma edited, but something like "Percent Disposed Value", the ratio of Paid Losses to Incurred Losses, will have a home-grown Percent format, PCTDP. Since we are using PROC PRINT and since we want to get the right format for each variate, we will produce reports for the items (variates) one at a time. Thus the report generation process will be in the form of a loop.

The code follows:

```
OPTIONS NOBYLINE PAGESIZE=66;
TITLE "RESERVING SYSTEMS FRONT END";
TITLE2 "SB0411RP ---- TRIANGLE REPORTS";
TITLE3 "AS OF &prodate.";
TITLE4 "#BYVAL(ITEMNAME)"
TITLE5 "#BYVAL(ARGUM)"
%do jj = 1 %to &novars;
  %if &allyears=YES
  %then
    let startyr=&firstyr.;
  %else
    let startyr=%eval(&proyear.-12);
  %let finalyr=proyear.;
  OPTIONS PAGENO=1;
  %do until(&finalyr LT &startyr);
    %let inityr=%eval(&finalyr.-12);
    %if &inityr LT &firstyr
    %then %let inityr = &firstyr.;
    PROC PRINT DATA=PRNTFILE label;
      WHERE ITEMNAME  = "&&&vname&jj.";
      BY LINE SUBL ARGUM ITEMNAME;
      PAGEBY ITEMNAME;
      ID JDMO;
      VAR AYR&inityr.-AYR&finalyr.
      FORMAT ITEMNAME $VLABELC.
      AYR&inityr.-AYR&finalyr. $&format&jj.;
    RUN;
    %let finalyr = %eval(&finalyr.-13);
  %end;
%end;
```
The outer %DO loop exercising PROC PRINT generates the reports by variate, one after the other. The inner loop iterates by groups of accident years. Taken together, both illustrate a further application of the use of indexed lists of macro variables. By doing one variable at a time, the appropriate format for the variate can be picked up from the indexed list of SAS Formats, &format1, &format2, &format3, etc. See the FORMAT statement in the PROC PRINT step.

The $VLABELC format used to print ITEMNAME is yet another format actually derived from the VARTABLE dataset. The argument this time, however, is the character ITEMNAME holder of the SAS variable name, not the item number VARNUMB.

The use of BYVAL feature in the TITLE's permits the variable values of the ITEMNAME and the line of business title, LINENAME, to be represented in the title lines in a manner dictated by the SAS formats they respectively carry on the data file. LINENAME, you may recall is a four-digit numeric that has a character format, the line of business title.

12.0 Creating Output Data Files

Getting data into spreadsheets from SAS is an area where new technologies have created new and spiffier ways to bridge the application gaps -- for example, ODBC in the WINDOWS environment.

The present application actually runs on a DEC Alpha mid-tier under the Alpha's operating system. The Alpha also supports a PATHWORKS® network, a LAN arrangement, whereby the DEC's file management and printer facilities are available from the WINDOWS 3.1 or WINDOWS 95 PC environment.

Unfortunately, many of our PC users do not presently have access to PC/SAS, and they are therefore not as yet in a position to benefit from ODBC. For this reason, the application uses an older strategy to get data to the spreadsheet users: it produces flat files -- ".dat" files -- that can be easily and quickly imported into LOTUS® or EXCEL®. These are, in fact, the kind of file that can be "slash-file-import-numbers" imported into LOTUS.

This means:
   ◦ The fields are blank-separated; and
   ◦ The character string (non-numeric) fields are quoted with double quotes.

The SAS code follows. The opening %IF ensures that a ".dat" file will only be produced if the user requested it:

```sas
%if &outfile ne %then %do;
   DATA CNTLFILE
     (KEEP = FMTNAMT START LABEL);
   SET RESRVSRC.VARTABLE
     FMTNAME = "$FACTOR"
     START = ITEMNAME;
   IF SUBST(VFORMAT,1,5)="COMMA"
     THEN LABEL = "NO";
   ELSE LABEL = "YES";
   OUTPUT;
   RETURN
RUN;
   PROC FORMAT CNTLIN=CNTLFILE;
RUN;
   DATA _NULL_;
      ARRAY ADATA(&firstyr.:&proyear.)
         AYR&firstyr.-AYR&proyear.;
      LENGTH LINECHR $5   FACTOR $ 3 ;
      LENGTH NAMECHR $35;
      SET RESRVSRC.TRIANGLE;
      BY LINE SUBL ITEMNAME;
      FILE 
         "&outfile..PRN" LRECL=500;
      IF FIRST.ITEMNAME THEN DO;
         LINECHR = PUT(LINE,Z2.)
            || ' -'
            || PUT(SUBL,Z2.);
         NAMECHR = PUT(ARGUM,SUBLNAME.);
         PUT /@1 '"' LINECHR $SCHAR5. '"'
            @2 '"' NAMECHR $ '';
         PUT /@1 '"' ITEMNAME $VLABELC. '"';
         PUT /@1 '"DEV'' @;
         DO JJ = LBOUND(ADATA) TO HBOUND(ADATA);
            PUT +2 JJ Z4. @;
         END;
         PUT ' '; 
         FACTOR = PUT(ITEMNAME,$FACTOR.)
            END;
         PUT @1 JDMO 5. @;
         DO JJ=LBOUND(ADATA) TO HBOUND(ADATA);
            IF FACTOR = 'YES'
               THEN PUT +1 ADATA(JJ) 9.5 @;
               ELSE PUT +1 ADATA(JJ) 9. @;
         END;
         PUT ' ';
         RETURN;
RUN;
%end;
```

The '"' business ensures the output character data will be double quoted. Note the use of variable extents in the array ADATA.

To appropriately represent the variate on the file as a number or factor, we need to know the format of the item. This time we generate the format on the fly from VARTABLE. (The permanent formats referenced earlier were generated in the same manner.)
13.0 And In Closing ...

Lest we forget, the macro ends -- and with it our walk-through -- with:

%mend YR0141PC;

As noted at the beginning, the application covers many of the basic features in the macro language. The application also makes an elementary use of SAS/AF (to access the application) and SAS/FSP to update tables the application uses, like VARTABLE.

Some of the usage of macro variables -- specifically in extents and arrays -- is driven by the nature of SAS arrays as opposed to arrays in a language like PL/1. In PL/1, one could actually code the array extents as variables ("controlled storage") and set them up at execution time.

The use of "meta-data" is much in vogue as an aid in managing applications, in large part due to the new push to "data warehousing". But some degree of "meta-data" has always been part of SAS -- it has been a key product strength of SAS -- and indeed we have leveraged off SAS's variable labels and formats here. The triangle builder has taken the "meta-data" one notch further: entries on the table define computations in the application.

As of this writing, the triangle builder has already had one major extension: to accident quarter, as opposed to accident year, data. It’s alive and growing.

Acknowledgements:

SAS® is a registered trademark of the SAS Institute, Inc., of Cary, North Carolina.

LOTUS® is a registered trademark of the Lotus Development Corporation.

EXCEL® is a registered trademark of Microsoft, Inc.

PATHWORKS® is a registered trademark of Digital Equipment Corporation, Inc.

The SAS macro language is covered in depth in the SAS Macro Language Reference, First edition, published by the SAS Institute, Inc.

My thanks to fellow Hartford employees Sheri Barnicle, Peter Prause and Chuck Patridge, who reviewed this paper and gave invaluable feedback.