Paper 267-29

SOME USES (AND HANDY ABUSES) OF PROC TRANSPOSE

Ralph W. Leighton, The Hartford

1.0 OVERVIEW:

PROC TRANSPOSE is a powerful – and somewhat underutilized -- data manipulation tool in Base SAS®. Using it, one can switch the significance of row and column identifiers, either globally or selectively within BY-groups. That is to say, it permits one to switch SAS Variables (columns) with logical record keys (rows). This capability is handy, if (for example) you use financial data that has a time period like "year" as a key (class variable) and you want to display the data down the page with the years as columns. (We'll use such an example shortly.)

After introducing PROC TRANSPOSE using an example that makes use of all the procedure's options, the bulk of the remainder of this paper covers some applications that may at first glance seem outside of the basic purpose of PROC TRANSPOSE. These topics include:

- Rotated Reports with Varying formats;
- Reports with uniform blocking;
- SAS Labels for Indexed Lists; and
- Record-to-record arithmetic

Although seemingly a disparate set of considerations or problems, at least three are not unrelated. First they suggest a general application (a SAS macro) for producing a class of landscape reports, a topic covered in a shorter paper by this author in the NESUG 1998 Proceedings. They also provide the foundation of a generic processing solution to a key problem in Casualty Insurance: the transformation of Claim (Loss) accounting data into so-called **loss development triangles**, a key type of display used by Reserving and Pricing Actuaries in the industry. Section 7.0 examines this application in some detail.

2.0 A 50-CENT INTRODUCTORY TUTORIAL:

We begin with a basic example of the PROC TRANSPOSE procedure for those readers not acquainted with the procedure.

Suppose I have a SAS data set **MYFIRM.SALEDAT1** described in the box below. The nature of this data is illustrated in the PROC PRINT "Report 1.0", shown in part on the next page.

```
============= SOURCE DATASET "SALEDAT1" ==============
            Variable    SAS-Format     SAS-Label

KEYS        PRODUCT     $CHAR12.       Line-of Business
            RSTATE      $CHAR2.        State
            CALYEAR     4.             Calen Year

VARIATES    COMMISNS    COMMA9.        Salesman Commission
            COSTS       COMMA9.        Other Costs
            PROFIT      COMMA9.        Profit fr-Sales
            SALES       COMMA9.        Sales Revenue
            STAXES      COMMA9.        State Sales-Tax
            TOPMAN      $CHAR9.        Yrs-Best Salesman
```

If you browse that PROC PRINT dump of the dataset, you will observe a few features embodied in the data:

- The data relates to sales of different items in different states in different calendar years;
- Not all products were sold in every state in every year. For example, auto parts were not sold in Connecticut in 1998 (presumably business was discontinued there). And garden tools were not sold in Massachusetts until 1997.

A different view of the same data would be to display the data items (Sales Revenue, Taxes and so on) as ***rows*** and the calendar year values as ***columns***. This form of display is very popular with financial analysts, who often like to look for trends across accounting periods. For those acquainted with SAS arrays, the data rearrangement can be accomplished with that tool, and this paper will touch on that approach later on, in Section 6.0 (Row Arithmetic). But an alternative solution path – and a non-procedural one at that -- is to use PROC TRANSPOSE, and the SAS code shown on the next page uses PROC TRANPOSE to accomplish the desired rotation of the data.

ACME ASSORTED SUPPLIES CORPORATION 1
REPORT 1.0
INCOME SUMMARY [File SALEDA1]

Line-of Business	State	Calen Year	Sales Revenue	State Sales-Tax	Salesman Commission	Other Costs	Profit fr-Sales	Yrs-Best Salesman
Auto-Parts	CT	1996	329,746	16,990	39,637	206,094	67,024	Nerwell
		1997	281,288	15,731	37,936	182,300	45,322	Fargo
Auto-Parts	MA	1996	375,116	17,278	73,877	269,538	14,422	Lisowski
		1997	291,477	14,572	64,406	203,616	8,883	Frederics
		1998	331,444	14,499	60,421	245,161	11,364	Lisowski
Auto-Parts	NY	1997	1,282,076	90,290	101,324	911,082	179,380	Nerwell
		1998	1,695,572	104,484	110,553	1,213,647	266,888	Hovgaard
Garden-Tools	CT	1996	606,320	32,975	48,164	444,437	80,745	Frederics
		1997	502,154	29,641	44,754	374,750	53,009	Patridge
		1998	608,611	31,434	44,750	460,150	72,278	Nerwell
Garden-Tools	MA	1997	194,527	22,402	75,509	70,487	26,129	Frederics
		1998	214,758	21,640	68,773	91,891	32,453	Lisowski
Pool-Tables	CT	1998	484,769	66,210	28,693	243,276	146,590	Davis
Pool-Tables	NY	1996	731,877	66,412	88,628	469,612	107,224	Hovgaard
		1997	622,884	61,346	84,628	404,572	72,337	Hovgaard
		1998	775,790	66,855	86,957	520,621	101,357	Patridge

```
PROC TRANSPOSE    DATA = MYFIRM.SALEDAT1
                    OUT = MYFIRM.ROTATED1
                 PREFIX = CYR
                   NAME = ORIGVAR
                  LABEL = ITEMNAME
      ID          CALYEAR;
      BY          PRODUCT  RSTATE;
      VAR         SALES    PROFIT   STAXES   COMMISNS COSTS;
      IDLABEL     CALYEAR;
RUN;

PROC PRINT DATA=MYFIRM.ROTATED1 LABEL;
  BY       PRODUCT RSTATE;
  ID       PRODUCT RSTATE;
  VAR      ITEMNAME  ORIGVAR  CYR1996-CYR1998;
  LABEL  ORIGVAR  = "Original Variable"
         ITEMNAME ="Income Item ..";
RUN;
```

In the above SAS Code, as is common with many SAS Procedures, the DATA= option identifies the input data set **SALEDAT1**, and the OUT= option identifies the output file **ROTATED1** (with the rotated data) created by the procedure. The rest of the syntax is as follows:

BY-statement: This works as elsewhere in SAS -- the rotations will take place on data within the lowest level sort-break of the BY-list specified – that is to say with every change of Reporting State, **RSTATE**. The BY variables **PRODUCT** and **RSTATE** will be on the output SAS Dataset.

VAR-statement: This identifies the data items (variables or columns) selected for rotation. Variables not listed in the BY statement or this list are dropped. In the present example, character variable **TOPMAN** is not in the list and therefore will be dropped. Why we have excluded it for the moment will be apparent later on, when we talk about mixed data formats in Section 4.0.

ID-Statement: The ID statement identifies a variable whose values will supply the SAS names for variables on the rotated data **ROTATED1**. The columns are to represent Calendar years, and indeed **`ID CALYEAR`** designates variable **CALYEAR** to supply calendar years, a four digit number. By themselves, these values cannot be SAS Variable names, and so read about the next option.

PREFIX= option: Since calendar year values are numeric and SAS variables cannot begin with a numeric, we use the PREFIX= option to prefix each numeric year with "**CYR**" (`PREFIX = CYR`). Our output data set will thus contain the indexed list **CYR1996-CYR1998.** (If you don't specify a prefix here, SAS will, using "**COL**".) If the ID variable chosen were character, we would not have to use this option.

IDLABEL-statement: The IDLABEL statement is used to designate a SAS variable whose row values will supply SAS Labels to the new variables (columns) on the output dataset **ROTATED1**. In this case the new variables will be **CYR1996**, etc. A natural label for a report is the year value itself, and hence the choice: `IDLABEL CALYEAR`. (In the next section, we will see an example where the choice of the IDLABEL variable will be different than that selected for ID.)

LABEL= option: The LABEL= option designates a user defined SAS Variable (character) on the Output file to hold the SAS Labels of the original variables (those in the VAR list). If your source data has the virtue of having SAS Labels, these will now supply descriptive row identifiers for reports made on the rotated data. Our data file **SALEDAT1** does have SAS Labels – see the "contents" listing on the first page of this paper -- and thus these SAS Labels will be saved in a variable called **ITEMNAME**. For example, **COMMISSNS** in the original file **SALEDAT1** and also on the VAR list has the SAS Label "Salesman Commissions". **ITEMNAME** will now carry this label as a value in rows occupied by the sales commission data, on the rotated version of the file, dataset **ROTATED1**.

NAME= option: Similarly, the NAME= option designates a user defined SAS Variable (character) on the Output file **ROTATED1**, which will hold original SAS Variable Name. For reference here (and use later), we elect to store these names in character variable **ORIGVAR**. For example, in the rows holding the sales commission data, **ORIGVAR** will have the value "COMMISNS". Why we might want this may not be immediately apparent, since it seems to add nothing to the landscape report from the standpoint of the report's business value. This option will reveal its importance later on.

The PROC PRINT of the output SAS Dataset **MYFIRM.ROTATED1**, itself created by PROC TRANSPOSE, produces Report 2.0 (shown in below). You should note the formatting of the data: The original SAS variables in **SALESDAT1** all had the COMMA numeric SAS Format, and the rotation gave these COMMA formats to the new column variables, so that formats for these did not have to be defined on the output file.

ACME ASSORTED SUPPLIES CORPORATION

1

Report 2.0

INCOME SUMMARY - ROTATED VERSION [File ROTATED1]- SHOWING YEARS

Line-of Business	State	Original Variable	Income Item Descrip	1996	1997	1998
Auto-Parts	CT	SALES	Sales Revenue	329,746	281,288	.
		PROFIT	Profit fr-Sales	67,024	45,322	.
		STAXES	State Sales-Tax	16,990	15,731	.
		COMMISNS	Salesman Commission	39,637	37,936	.
		COSTS	Other Costs	206,094	182,300	.
Auto-Parts	MA	SALES	Sales Revenue	375,116	291,477	331,444
		PROFIT	Profit fr-Sales	14,422	8,883	11,364
		STAXES	State Sales-Tax	17,278	14,572	14,499
		COMMISNS	Salesman Commission	73,877	64,406	60,421
		COSTS	Other Costs	269,538	203,616	245,161
Auto-Parts	NY	SALES	Sales Revenue	.	1,282,076	1,695,572
		PROFIT	Profit fr-Sales	.	179,380	266,888
		STAXES	State Sales-Tax	.	90,290	104,484
		COMMISNS	Salesman Commission	.	101,324	110,553
		COSTS	Other Costs	.	911,082	1,213,647

Also take note of the missing values for Connecticut Auto Parts in 1998 and New York Auto parts in 1996. Remember, there were no source records for these combinations on the source file **SALEDAT1**. Thus, when the SAS dataset was rotated, a missing values entry was created, since something has to occupy each row (record) instance of a SAS variable (column). The same thing would occur if you used arrays to effect the same data rotation.

3.0 THE DOUBLE TRANSPOSE ("FLIP-FLOP"):

If one transposes the rotated data set back again, one gets the original data set -- well, almost. The following code will effect this restoration:

```
PROC TRANSPOSE  DATA=MYFIRM.ROTATED1
                 OUT=WORKFIL1
               LABEL=CYRCH;
   ID       ORIGVAR;
   BY       PRODUCT  RSTATE;
   VAR      CYR1996-CYR1998;
   IDLABEL ITEMNAME;
RUN;

DATA MYFIRM.SALEDAT2;
   ATTRIB CALYEAR  FORMAT=4. LABEL="Calen Year";
   SET WORKFIL1;
   CALYEAR = CYRCH;
RUN;

PROC PRINT DATA=MYFIRM.ROTATED1 LABEL;
  BY       PRODUCT  RSTATE;
  ID       PRODUCT  RSTATE;
  VAR      CALYEAR  SALES   PROFIT   TAXES   COMMISNS  COSTS;
RUN;
```

A fragment of the report appears below (Report 3.0). The ID statement in the PROC TRANSPOSE execution references the **ORIGVAR** variable on dataset **ROTATED1** to pick up the names of the original numeric variables. The IDLABEL statement references **ITEMNAME** to pick up the original SAS Labels, which again show as labels to the columns. Note the following about the report

- Missing values entry for Connecticut Auto Parts in 1998. On the original file SALESDAT, there was no such entry, since there were no sales of Auto Parts in Connecticut for 1998. The double rotation has had the effect of "filling in" (with null records) entries for missing combinations on the source data. In those odd instances where a user demands a report with uniform blocking of the data (for readability or whatever), the **double rotation gimmick** will often provide a solution path.
- Recovery of **CALYEAR** as a numeric variable. To do this I had to add a DATA-step after the PROC TRANSPOSE execution. Remember that the values of **CALYEAR** are in the SAS Labels for the indexed list CYR1996-CYR1998 in the transposed data set **ROTATED1**. By using the LABEL= option on the rotation back, we can recover the values, and option LABEL-CYRCH means that variable CYRCH will contain those values. Unfortunately, the target of the LABEL= option is always a character variable, even though in this instance all the label values happen to be numeric. The follow on DATA-step loads the values from character variable **CYRCH** into the numeric **CALYEAR**.
- Absence of **TOPMAN** (the sales person name) on this report. It was not one of the variables we transposed on the first rotation (that is to say, it was not on the VAR list). We'll come back to this shortly.

```
                        ACME ASSORTED SUPPLIES CORPORATION                    1
                                    Report 3.0
                  INCOME SUMMARY - RESULT OF DOUBLE ROTATION [File SALEDAT2]
Line-of                    Calen       Sales      Profit      State      Salesman       Other
Business        State      Year      Revenue    fr-Sales   Sales-Tax   Commission      Costs

Auto-Parts       CT        1996      329,746      67,024      16,990      39,637     206,094
                           1997      281,288      45,322      15,731      37,936     182,300
                           1998            .           .           .           .           .

Auto-Parts       MA        1996      375,116      14,422      17,278      73,877     269,538
                           1997      291,477       8,883      14,572      64,406     203,616
                           1998      331,444      11,364      14,499      60,421     245,161
                                                E T C .
```

4.0 TRANSPOSING MIXED FORMAT DATA:

What will happen, if we rotate numeric data with mixed SAS formats? To illustrate, let's add the following data elements to our original file.

- Commission Ratio: The percent of income doled out in sales person commissions.
- Sales Tax Ratio: similarly the ratio of Sales Taxes Paid out to Sales.
- The annual Percent Change in Sales

The following DATA step creates this augmented version **SALEDAT3** of the original **SALEDAT1** file:

```
DATA MYFIRM.SALEDAT3;
  ATTR  COMRATIO FORMAT=PERCENT7.1    LABEL="Commisn Ratio";
  ATTR  STXRATIO FORMAT=PERCENT7.1    LABEL="State Tax-Pct";
  ATTR  SALESCHG FORMAT=PERCENT7.1    LABEL="Pct-Chg in-Sales";
  ********************************************;
  SET MYFIRM.SALEDAT1;
  BY PRODUCT RSTATE;
  COMRATIO = COMMISNS/SALES;
  TAXRATIO = STAXES/SALES;
  SALESCHG = SALES/(LAG(SALES) - 1.0;
  IF FIRST.STATE THEN SALESCHG = .
RUN;
```

Now suppose that I choose to rotate (transpose) only these three new elements, using PROC TRANSPOSE, as per:

```
PROC TRANSPOSE   DATA = MYFIRM.SALEDAT1
                    OUT = MYFIRM.ROTATEDA
                 PREFIX = CYR
                   NAME = ORIGVAR
                  LABEL = ITEMNAME
      ID         CALYEAR;
      BY         PRODUCT   RSTATE;
      VAR        COMRATIO  TAXRATIO  SALESCHG;
      IDLABEL  CALYEAR;
RUN;

PROC PRINT DATA=MYFIRM.ROTATEDA LABEL;
  BY      PRODUCT RSTATE;
  ID      PRODUCT RSTATE;
  VAR     ITEMNAME  ORIGVAR  CYR1996-CYR1998;
  LABEL  ORIGVAR  = "Original Variable"
         ITEMNAME ="Income Item ..";
RUN;
```

Then the PROC PRINT would show a nice rotated (landscape) display, similar to Report 2.0, but with the PERCENT formats preserved, just as the COMMA formats were in the initial example using the dollar items. All well and good, but a more interesting question is: what happens if I rotate **all** the variates on **SALEDAT3**, both the dollars and the percents, as per the following code:

```
PROC TRANSPOSE  DATA=MYFIRM.SALEDAT3
                   OUT=MYFIRM.ROTATED3
                PREFIX=CYR
                NAME=ORIGVAR
               LABEL=ITEMNAME;
   ID  CALYEAR;
   BY  PRODUCT RSTATE;
   VAR SALES   PCTCHGSL  COMMISNS  COMRATIO  PROFIT
       STAXES  STXRATIO  COSTS;
   IDLABEL CALYEAR;
RUN;

PROC PRINT DATA=MYFIRM.ROTATED3 LABEL;
  BY      PRODUCT RSTATE;
  ID      PRODUCT RSTATE;
  VAR     ITEMNAME  ORIGVAR  CYR1996-CYR1998;
  LABEL  ORIGVAR  = "Original Variable"
         ITEMNAME ="Income Item ..";
RUN;
```

PROC PRINT of the resulting **ROTATED3** is shown on the top of the next page as Report 4.0. The formatting of the numeric data leaves something to be desired, and the reason is as follows: With the source variables in **SALEDAT3** having mixed formats, the resulting rows in the output data sets would have to have different formats, some rows percentages, other rows comma edited numbers. But a SAS variable cannot have different default formats on different records. Thus, the variables **CYR1996-CYR1998** will have single SAS format, "general", an attempt by SAS to accommodate the different orders of magnitude. SAS is making the best of a tough situation.

ACME ASSORTED SUPPLIES CORPORATION
Report 4.0
INCOME AND RATIO SUMMARY - DATA ACROSS YEARS [File ROTATED3]

Line-of Business	State	Original Variable	Income Item Descrip	1996	1997	1998
Auto-Parts	CT	SALES	Sales Revenue	329745.58	281288.15	.
		PCTCHGSL	Pct-Chg in-Sales	.	-0.15	.
		COMMISNS	Salesman Commission	39636.95	37935.57	.
		COMRATIO	Commisn Ratio	0.32	0.38	.
		PROFIT	Profit fr-Sales	67024.40	45321.56	.
		STAXES	State Sales-Tax	16990.31	15730.77	.
		STXRATIO	State Tax-Pct	0.05	0.06	.
		COSTS	Other Costs	206093.93	182300.25	.
Auto-Parts	MA	SALES	Sales Revenue	375115.60	291476.83	331444.13
		PCTCHGSL	Pct-Chg in-Sales	.	-0.22	0.14
		COMMISNS	Salesman Commission	73877.46	64405.78	60420.61
		COMRATIO	Commisn Ratio	0.70	0.73	0.70
		PROFIT	Profit fr-Sales	14421.67	8882.88	11363.52
		STAXES	State Sales-Tax	17278.20	14571.81	14498.66
		STXRATIO	State Tax-Pct	0.05	0.05	0.04
		COSTS	Other Costs	269538.27	203616.35	245161.34

E T C .

Clearly one can get around the formatting issue by creating a DATA-step report from the transposed file instead of using PROC PRINT. There is nothing wrong with this solution path. However, DATA-step reports involve some procedural code, custom to the file being printed, to space out the data nicely in columns and a "header" subroutine to supply the column headers with each page break. And if you have to make a lot of these reports, this kind of coding can get pretty tedious.

Oddly enough, an alternative solution path to a nicer looking report is suggested by what happens when we add the **TOPMAN** sales person name code to the variables being transposed:

```
PROC TRANSPOSE  DATA=MYFIRM.SALEDAT3
                 OUT=MYFIRM.NICEFILE
              PREFIX=CYR
                NAME=ORIGVAR
               LABEL=ITEMNAME;
  ID  CALYEAR;
  BY  PRODUCT RSTATE;
  VAR SALES   PCTCHGSL  COMMISNS  COMRATIO  PROFIT
      STAXES  STXRATIO  COSTS     TOPMAN ;
  IDLABEL CALYEAR;
RUN;
```

If you PROC PRINT this file, you will get the Report 5.0 shown at the top of the next page. Mysteriously, the nice formatting has reappeared. And the question is "**Why**?" A clue lies in a note in the SAS Log under the PROC TRANSPOSE execution.

```
NOTE: Numeric variables in the input data set will be converted to character in
      the output data set.
```

And, indeed, if you run this piece of code and check a PROC CONTENTS of the file NICEFILE, you will see that, indeed, the output indexed list CYR1996-CYR1998 consists of character string variables. When SAS rotated (i.e. transposed) only numeric variables, the resulting variables **CYR1996-CYR1998** on the output file could be (and were) numeric. But, when SAS is faced with rotating a collection of variables of mixed data type, it cannot create something that is character on some records and numeric on others. Thus SAS chooses the character data type.

O.K, so what? Look carefully at Report 5.0: What did SAS do to the values of the original numeric variables when it made the output character? SAS did a rather nice thing: it transferred the rotated values of the source variables onto the output file character strings **CYR1996-CYR1998** *using, in each instance, the source variable's SAS Format*. (Recall: the original dollar amounts had COMMA formats and the added percentages were given PERCENT formats.) SAS also took some pains to line up the data so the numbers have consistent representation in the columns. And finally, look at the order of the row instances:

they are precisely the order in which the variables were listed in the VAR-List. in the PROC TRANSPOSE code

```
                        ACME ASSORTED SUPPLIES CORPORATION                          1
                                     Report 5.0
                  INCOME AND RATIO SUMMARY - DATA ACROSS YEARS [File NICEFILE]

Line-of                  Original
Business        State    Variable    Income Item Descrip    1996          1997        1998

Auto-Parts       CT      SALES       Sales Revenue           329,746      281,288
                         PCTCHGSL    Pct-Chg in-Sales              .     (14.7%)
                         COMMISNS    Salesman Commission      39,637       37,936
                         COMRATIO    Commisn Ratio            32.1%        38.3%
                         PROFIT      Profit fr-Sales          67,024       45,322
                         STAXES      State Sales-Tax          16,990       15,731
                         STXRATIO    State Tax-Pct             5.2%         5.6%
                         COSTS       Other Costs             206,094      182,300
                         TOPMAN      Yrs-Best Salesman      Nerwell      Fargo

Auto-Parts       MA      SALES       Sales Revenue           375,116      291,477      331,444
                         PCTCHGSL    Pct-Chg in-Sales              .     (22.3%)       13.7%
                         COMMISNS    Salesman Commission      73,877       64,406       60,421
                         COMRATIO    Commisn Ratio            70.0%        73.3%        70.0%
                         PROFIT      Profit fr-Sales          14,422        8,883       11,364
                         STAXES      State Sales-Tax          17,278       14,572       14,499
                         STXRATIO    State Tax-Pct             4.6%         5.0%         4.4%
                         COSTS       Other Costs             269,538      203,616      245,161
                         TOPMAN      Yrs-Best Salesman       Lisowski     Frederics    Lisowski

Auto-Parts       NY      SALES       Sales Revenue                      1,282,076    1,695,572
                         PCTCHGSL    Pct-Chg in-Sales                           .         32.3%
                         COMMISNS    Salesman Commission                  101,324      110,553
```

All this suggests a ploy some might call a “Stupid SAS Trick”. Suppose you have a data set **SOURCE** with mixed formats (like the file **SALEDAT3**). Suppose you want a transposed report like the one below, but don’t happen to have a character variable to stuff into the VAR list. What to do to get all that nice formatting on the rotated version of the data? Simply create an intermediate file **TEMPFILE** with a character variable, like **CRAP** (below):

```
DATA TEMPFILE;
    SET MYFIRM.SOURCE;
    CRAP="abc";
```

Now transpose this temporary file, making sure that **CRAP** appears in the VAR-list.

```
PROC TRANSPOSE  DATA=TEMPFILE;
                 OUT=PRNTFILE
                NAME=ORIGVAR
          <<etc.>>
             ::
    VAR      ::::::::::<etc>  CRAP;
```

Then, in the PROC PRINT (or PROC REPORT) step, simply use a WHERE statement to drop the added character value.

```
PROC PRINT DATA=PRNTFILE LABEL;
      WHERE ORIGVAR NE "CRAP";
             ::
             ::
RUN;
```

In short, “add a little crap” to your data, and then do remember to remove it at report time. This section now concludes with three observations regarding the transposing (rotation) of data, in particular those involving mixed formats.

First of all, it is hoped the reader will see advantages to having SAS Formats and SAS labels associated with the variables on SAS datasets one uses. The presence of these has clearly been taken advantage of in the examples so far discussed. Their absence would have meant more code to format the data nicely on the reports.

Secondly, in the last example (shown on Report 4.0) there were a couple of subtle changes in formatting from that appearing

on Report 2.0. First of all, the dummy instances created for non-existent source records on the original file **SALEDAT3** --e.g. Connecticut Auto-Parts in 1998 -- no longer contain missing values but rather contain blanks. This is because SAS formats the data to character first and then rotates it, and the "missing value" for a character variable is blanks. Also, the Calendar Year label is not longer centred but rather is left justified over the columns, since the **CYR** variables are character. You can fix both of these problems.

The formatting of the calendar year label can be rectified by giving **CALYEAR** a wider SAS format in the DATA step creating the intermediate file **TEMPFILE** where variable **CRAP** was added to the data:

```
FORMAT CALYEAR 7.;
```

The larger SAS format will cause the year labels above the **CYR** variables to move over to the right 3 spaces on the transposed data file.

Replacing the "Missing value" blanks in the report data areas entails a tiny bit of additional code. One way to do that is to have SAS dynamically replace blanks with something – say "n/a" for "Not Applicable". – and this can be done by creating a little dumb VALUE format FILLER:

```
PROC FORMAT
     VALUE $FILLER  " " = "...n/a...";
Run;
```

If you do this, be careful about the width of the replacement string on the right side of the assignment: its length defines the width of the character variables printed with the format, whether or not the substitution takes place. It the example above the replacement has been carefully set to span **9** characters, the width the variables **CYR1996-CYR1998** happened to be, since the largest rotated value is occupied by numbers printed using COMMA9 format. Although I have used dots (periods) as filler in defining the blank-replacement character string for $FILLER, you can use blanks as well. Once the temporary format $FILLER is set up, use it in the PROC PRINT of the transposed file by adding the line below to the code:

```
FORMAT  CYR1996-CYR1999   $FILLER. ;
```

The result of these two cosmetic changes is illustrated by the revised version of Report 5.0 below:

```
                          ACME ASSORTED SUPPLIES CORPORATION                    1
                                    Report 5.0 - A
        INCOME AND RATIO SUMMARY - DATA ACROSS YEARS - [file NICEFILE] - Cleaned Up

Line-of                 Original
Business        State   Variable   Income Item Descrip      1996        1997        1998

Auto-Parts       CT     SALES      Sales Revenue           329,746     281,288    ...n/a...
                        PCTCHGSL   Pct-Chg in-Sales            .      (14.7%)    ...n/a...
                        COMMISNS   Salesman Commission      39,637      37,936    ...n/a...
                        COMRATIO   Commisn Ratio            32.1%       38.3%     ...n/a...
                        PROFIT     Profit fr-Sales          67,024      45,322    ...n/a...
                        STAXES     State Sales-Tax          16,990      15,731    ...n/a...
                        STXRATIO   State Tax-Pct             5.2%        5.6%     ...n/a...
                        COSTS      Other Costs             206,094     182,300    ...n/a...
                        TOPMAN     Yrs-Best Salesman     Nerwell     Fargo        ...n/a...

                                        E T C .
```

The third and final observation about mixed format data has to do with the intended target use of the transposed mixed format data. What went into Report 5.0 looks very pretty, but it is really not very amenable to arithmetic manipulation: the variables are character. For computational use, the original output **ROTATED3** that fed the marginal looking Report 4.0 (with numeric versions of **CYR1996-CYR1998**) is what you want. Thus whether or not you "throw a little crap" in with your data, so to speak, depends on what you are intend to do with the transposed output.

5.0 LABELS FOR INDEXED LISTS:

Suppose you have an indexed list of, say, numeric variables, like: **ACMO1 - ACMO256** and you want to assign SAS Labels to them, like "Acc Mon –27-" for ACMO27. You have some choices in the way to go about this:

- **"The Method of Exhaustion"**: You can write out 256 LABEL statements in your SAS code .

- **Macro solution**. You could code a little macro that will generate 256 LABEL statements and then invoke the macro in SAS Code, like a DATA Step or PROC DATASETS. But If the SAS Code in question is already part of a macro, then you can use a macro %do loop in the macro to produce the labels as per that below which gives, for example, the SAS Label "Acc-Mon --123---" to the 123rd entry:

```
%do jj=1 %to 256;
   LABEL ACMO&jj="Acc-Mon --&jj--";
%end;
```

- **PROC TRANSPOSE**: This is another one of those funny little off-beat applications, and one that is useful if writing SAS macros is not your bag.

Now, the third alternative does not entail rotating the data file you want to apply the SAS Labels to. Rather, it involves creating a dummy data set with the indexed list above, complete with SAS Labels, as per:

```
DATA DUMMY01
       (KEEP=JJ AMOUNT LABELV);;
    AMOUNT=0;
    DO JJ = 1 TO 256;
        LABELV = "Acc-Mon --" || LEFT(PUT(JJ,3.))||'--';
        OUTPUT;
        END;
RUN;

PROC TRANSPOSE  DATA=DUMMY01
                  OUT=DUMMY02 (DROP=_NAME_)
                PREFIX=ACMO;
     ID        JJ
     IDLABEL   LABELV;
     VAR       AMOUNT;
RUN;

DATA DUMMY03;
     SET DUMMY02;
     STOP;
RUN;
```

Note that this last file **DUMMY03** has no observations, but it does have variables **ACMO1-ACMO256** with SAS Labels. Now use a DATA step to append the null dataset **DUMMY03** to the data set needing the SAS Labels for the indexed list:

```
DATA <<sourcefile>>;
     SET  <<sourcefile>>
          DUMMY03 ;
```

The coding of this solution is clearly longer than, at least, the %do-loop shown in the second option. Yet, as noted, such a solution is an out for those not wishing to mess with SAS Macro code. And the PROC TRANSPOSE approach does seem to have an advantage in instances where several unrelated SAS datasets need labels for the same indexed list. The file **DUMMY03** can be created once as a permanent SAS Dataset, and then used multiple times.

A downside to the solution is the requirement that a DATA step (as per above) be the vehicle to bring in the labels from **DUMMY03**. This means the dataset **<<sourcefile>>** must be read through. A thought might be to bypass this processing by using PROC APPEND to add the null file **DUMMY03** to the end of the data **<<sourcefile>>**. If you try that althernative, you will find that the SAS Labels will not be picked up. On the other hand, in the SET statement the DATA step bringing in **DUMMY03**, the order of appearance of the two files **<sourcefile>>** and **DUMMY03** doesn't matter. Since presumably **<<sourcefile>>** does not have SAS labels (the reason you would be doing this in the first place) the output file always picks them up from **DUMMY03**.

6.0 ROW ARITHMETIC - OR "FLIP-FLOP" REVISITED

The term "row arithmetic" refers to situations wherein it becomes necessary to create observations as arithmetic combinations of other observations, usually within some BY-group processing.

The concept can be illustrated in the context of one of our earlier examples. Suppose you were given, not file **SALEDAT1**, but rather the file **ROTATED1** (cf. section 2.0) and were asked to calculate the percentage items **COMRATIO**, **STXRATIO**, and **SALESCHG**, as additional records (in this case) yielding the file which we called **ROTATED3** in section 4.0.

This subject formed part of a discussion in a NESUG 1998 Beginning Tutorial on the subject on the use of PROC TRANSPOSE versus the use of DATA steps, and the paper's author advocated using DATA step solutions to such problems.

Depending on the situation, such a DATA-step solution can be appropriate. But, as we shall now see, it can have some downsides. So we begin by illustrating how a DATA-step could form SAS Dataset **ROTATED3** from **ROTATED1**:

```
DATA MYFIRM.ROTATED3;
  ARRAY  ITEMS(*)    CYR1996-CYR1998;
  ARRAY  HLDSALES(*)  _TEMPORARY_;
  ARRAY  HLDCOMMS(*)  _TEMPORARY_;
  ARRAY  HLDSTAX(*)   _TEMPORARY_;
  ******************************;
  SET   MYFIRM.ROTATED1;
  BY    PRODUCT RSTATE;
  OUTPUT;
  DO JJ = 1 TO DIM(ITEMS);
    SELECT(ORIGVAR);
      WHEN("SALES")     HLDSALES(JJ)=ITEMS(JJ);
      WHEN("COMMISNS") HLDCOMMS(JJ)=ITEMS(JJ);
      WHEN("STAXES")    HLDCOMMS(JJ)=ITEMS(JJ);
      OTHERWISE;
      END;
    END;
  ******************************;
  IF ORIGVAR="SALES" THEN DO;
    DO JJ=DIM(ITEMS) TO 2 BY -1;
       ITEMS(JJ) =  ITEMS(JJ) / ITEMS(JJ-1);
       END;
    ITEM(1)   = .;
    ORIGVAR   = 'SALESCHG';
    ITEMNAME = "Pct-Chg in-Sales";
    OUTPUT;
    END;
  ******************************;
  IF LAST.RSTATE THEN DO;
    DO JJ=1 TO DIM(ITEMS);
      ITEMS(JJ) =  HLDCOMMS(JJ)/HLDSALES(JJ)-1;
      END;
    ORIGVAR  ="COMRATIO";
    ITEMNAME ="Commissn Ratio";
    OUTPUT;
    DO JJ=1 TO DIM(ITEMS);
      ITEMS(JJ) =  HLDSTAX(JJ)/HLDSALES(JJ)-1;
      END;
    ORIGVAR  = "STXRATIO";
    ITEMNAME = "State Tax-Pct";
    OUTPUT;
    END;
  ******************************;
RUN;
```

There's a fair amount of code here, and some justification for its presence is called for.

- There are three hold arrays (for Sales, Commissions and State Taxes). They are here because of uncertainty as to the order as to what comes first within each BY Group on **ROTATED1**. In the code above, source data is read through and output, and then, at the end of the **RSTATE** group, the additional items are calculated. (If we could be sure the "SALES" record came first, then we could dispense with the hold arrays for the commissions and the state taxes. In this case the two ratios could be calculated and output right after the dollar items were output, not at the end of the BY group.)
- There are a number of DO-loops to get the various additional items calculated. We avoid the use of specific index references by using the dimension of the input indexed data list **CYR1996-CYR1998**, which in this case is "3". The Percentage Sales Change avoids the use of an extra array by using the backwards-iterating loop shown. Note that the Percentage change for the first year 1996 is set to null.
- Although repetitions of ratios to sales might suggest some savings by using two dimensional arrays and double DO-loops, such a ploy here would not buy much code efficiency, since there are only two such items in this case.

An alternative approach to form **ROTATED3** is the **double-transpose ("flip-flop")**. In this three step approach, we first transpose the data in **ROTATED1** so that values in the item identifier (variable **ORIGVAR**) become SAS Variables -- i.e. **ORIGVAR** is the ID-variable. In the second step, a DATA step – and a short one at that -- calculates the derived items in the same man-

same manner as they were calculated in Section 4.0 -- i.e. using simple arithmetic (and no DO loops). Finally in the third step, a second transposition restores the data to its original configuration, as **ROTATED3**. The code for all three steps is below, and you will note that the second and third steps are little more than what was already presented in section 4.0:

```
PROC TRANSPOSE DATA=MYFIRM.ROTATED1
               OUT=WORKFILE1
                   (DROP=_NAME_)
              LABEL=CYRCHAR            /* See note below */
    ID         ORIGVAR;
    BY         PRODUCT  RSTATE;
    VAR        CYR1996-CYR1998;
    IDLABEL    ITEMNAME;
RUN;
*****************************;
DATA WORKFIL2;
  ATTRIB  COMRATIO FORMAT=PERCENT7.1          LABEL="Commisn Ratio";
  ATTRIB  STXRATIO FORMAT=PERCENT7.1          LABEL="State Tax-Pct";
  ATTRIB  SALESCHG FORMAT=PERCENT7.1          LABEL="Pct-Chg in-Sales";
  SET WORKFIL1;
  COMRATIO = COMMISNS/(SALES-COSTS);
  STXRATIO = STAXES / SALES;
  CTXRATIO = CTAXES / SALES;
  RETURN;
RUN;
*****************************;
PROC TRANSPOSE DATA=WORKFIL2
                 OUT=MYFIRM.ROTATED3
               PREFIX=CYR
                 NAME=ORIGVAR
                LABEL=ITEMNAME;
     ID     CYRCHAR;
     BY     PRODUCT  RSTATE;
   VAR SALES    PCTCHGSL  COMMISNS  COMRATIO  PROFIT
       STAXES   STXRATIO  COSTS;
     IDLABEL  CYRCHAR;
RUN;
```

If we wanted the output file **ROTATED3** simply for reporting purposes, we could have used the trick of "adding a little crap" to our data (cf. Section 4.0, Mixed Formats). The dummy character variable would be added as part of the DATA step code and added to the VAR-list in the second PROC TRANSPOSE. In this instance, the output **CYR** indexed list in **ROTATED3** would consist of character strings with formatted contents. In the presentation above, the SAS code yields **CYR** variables that are numeric, not character. Note also that I did not have to do anything about Calendar Year (**CYRCHAR**) being character, since I only need it except as an ID variable for the rotation back to "landscape" form.

It is worth comparing the two approaches.

- MAINTENANCE CONSIDERATIONS: The DATA-step approach is heavily procedural code and involves a fair number of indices and Do-loops; the program shown is 47 statements long. If you need to add more input data and calculated items to the application – as for example a Percent Profit (or Profit Ratio) -- this code will expand rather dramatically as additional loops and temporary storage arrangements (arrays) are set up. As alluded to above, a particular application might allow some code reduction by using two- or three- dimensional arrays. That was not done in the example for the reasons stated. While this kind of array usage might abbreviate code volume, it could also lead to headaches by obfuscating what is getting calculated. The result could be spiffier but less intelligible code.

  The TRANSPOSE approach seems to be easily adapted to enhancements. If one adds more source items (to **ROTATED1**) and/or further items to be calculated, only two small changes need occur: simple statements for new calculated items need be added in the DATA step; and all new items have to be added to the VAR-list in the second PROC TRANSPOSE. That is the total scope of the maintenance change.

- PROCESSING EFFICIENCY CONSIDERATIONS: On the other hand, the DATA step has an advantage of passing the data once, whereas the TRANSPOSE approach passes the same data three times. Clearly this is a consideration for any application involving very large record volumes. But, for an application involving moderate sized files, the number of passes should be a very much of a non-issue.

7.0 "FLIP-FLOP-FLAP" IN CASUALTY LOSS DEVELOPMENT TRIANGLE ANALYSIS

The final section of this paper will now apply some of the previously discussed material to a casualty actuarial data retrieval application, such as that at The Hartford: producing loss development triangle reports. First a bit of background.

Casualty insurance, as sold by Property-Casualty Insurers, tends to develop losses for some time -- sometimes for a very long time -- after the subject policies have expired (and the premium income long since collected). Even in a line like Personal Automobile, there may be a considerable lag from when an accident occurs (during the policy term) and when a bodily injury claim first appears and then later settles. Sometimes such an injury is subtle, not apparent, delaying the filing of a claim. Once filed, the claim enters a stage of being the subject of investigation and then possibly back-and-forth between the insurer and the claimant (or the claimant's attourney), before the claim is closed. Some claims go into suit in the courts. It is thus no surprise that the largest liability in such a business is the reserve for un-paid claims, these being of two classes:

- **Case Reserves**: those claims already reported but as yet unpaid.
- **I.B.N.R. Reserves (Incurrred but Not Reported):** claims yet to be reported to the company on past business written.

There is always uncertainty as to exactly how large this loss reserve is – particularly the I.B.N.R. piece -- and one of the actuarial functions is to evaluate claim patterns as a test for the adequacy (or redundancy) of the company's posted reserves. One important element in such analyses is the projection of loss development triangles, a miniature example of which appears in the figure below:

Accident Year Triangle-Report Layout
Incurred Losses
(Dollars in Thousands)

	←----------Accident Year----------→					
DEV-YR	1998	1999	2000	2001	2002	2003
1	23,456	26,781	29,637	31,567	36,442	35,113
2	31,221	35,871	37,902	39,011	41,312	.
3	33,784	38,112	40,587	43,144	.	.
4	34,901	39,341	42,021	.	.	.
5	35,231	40,552	.	.	.	.
6	35,454	.	.	.	.	.

Such a report shows, by the accident period of the losses (claims) – this case an aggregation to Accident Year -- the (usually) upward progression, by valuation date, of claim statistics such as shown in the table below, with SAS Variable names for the items that will be used later in SAS Code:

Property-Casualty Insurance Financial Reporting Variate	SAS Varname
Incurred Losses : Claim money amounts reported to the insurance company, as due to claimants. The aggregate is affected by new reported claims as well as changes in claim value arising from changes in estimate by adjusters and changes occurring at the point the claim is	**INCURRED**
Paid Losses : Claim money amounts actually paid out to claimants. For many lines of business this occurs at the point the claim is settled. However, in some lines, such as Workers Compensation, payments can occur without the claim closing.	**PAIDLOSS**
Created Claim Counts : Numbers of claims reported. The aggregate is affected primarily by new claims being reported. If one is looking at a particular sub-line of business, it can also be affected by claim coding errors that move a claim from one line of business to another.	**CREATCNT**
Settled Claim Counts : Number of claims closed with payment to the claimant. Claims that close at a count of +1. A previously settled claim that reopens adds a count of –1.	**SETLDCNT**
Cancelled Claim Counts : A previously open claim that closes for no payment to the claimant is called "cancelled" – usually because either the claim is not supported or because there is no coverage for it on the policy.	**CANCLCNT**
Paid Allocated Loss Expenses : Moneys paid out in the handling of claims. This means costs associated with the processing of a claim, such as the hiring of an adjuster to review a property loss, an attourney to act on the insurance company's behalf or medical staff hired to evaluation a claim or examine the claimant.	**PAIDALE**

Each entry in the Incurred loss triangle display represents the CUMULATIVE reporting so far on the accident year, first at the end of the accident year (Development Year 1), then 12 months later (Development Year 2) and so on at 12 month (one year) intervals. So for Accident year 2001, the cumulative reported incurred loss was $31,567 at the end of 2001. At the end of 2002, this figure had risen by $8,444 to $39,011, as of Development Year 2. At the end of 2003, the figure had risen again, by

$4,133 to the value $43,144 shown for Development Year 3. The older an Accident Year is, the more development periods can be shown (and are shown). Looking at Accident Year 1998, we see that this upward movement has slowed markedly. And indeed, the time series of the developments do eventually approach an asymptote, what the actuaries call the "ultimate" value of the accident period losses. If you can estimate what that ultimate value is – and actuaries are well paid to do so, using the development patterns inherent in the triangle display – then you can estimate the reserve needed for the accident period by subtracting from the "ultimate" the know reporting to-date. So if I think (or estimate) that Accident Year 2001 is going to hit ultimate at $45,000, the reserve needed to cover unreported 2001 losses as of 12/31/2003 is the $45,000 "ultimate" minus the $43,144 reported to date or $1,856. That is the number I now compare with what has actually been set aside.

The actuaries are interested in triangles of, not only basic reported items, the previously listed aggregates of items such as Incurred losses and created claim counts, but also items that can be derived from these basic items, such as

Property-Casualty Insurance Derived Triangle Variages	SAS Varname
Outstanding Reserves = Incurred Losses – Paid Losses	**OSRESERV**
Net Created Claims = Created Claims – Cancelled Claims	**NTCRTCNT**
Average Reported Claim = Incurred Losses / Created Claims	**AVEINCRD**
Percent Disposed Claims = (Settled Claims + Cancelled Claims) / Created Claims	**PCTDISPD**

The source data for triangle displays, such as the one on the preceding page, is often in the form of financial transactions. These carry calendar changes of the basic quantities, like paid losses, incurred losses and loss expenses. Such a source, shown summarized across claim contributions to Accident Year and Calendar Year, is shown below

Exhbt 1.0 Source Summarised Financial Loss Data:

ACC-YR	REPT-YR	DEV-YR	INCURRED	PAIDLOSS	…etc.	
::	::	::	::	::		
2000	2000	1	29637	::	::	| These
2000	2001	2	8265	::	::	| are the
2000	2002	3	2685	::	::	| annual
2000	2003	4	1434	::	::	| summaries
2001	2001	1	31567	::	::	| of the
2001	2002	2	8444	::	::	| financial
2001	2003	3	4133	::	::	| claim-level
2002	2002	1	26422	::	::	| transations
::	::	::		::	::	

To become a loss triangle the data must be transformed from calendar changes by processing date to the cumulative display of reportings by development period (year). Thus the delivery of data to the end-user (the actuary) involves several steps:

Step 1 : Calculation of Development Period : Based on the accident period and the reporting date. This has already been carried out in Exhibit 1.0 and appears as column "DEV-YR" on the above display of the source data in the figure. Development Year = Accident Year – Calendar Year + 1. One thing that also ought to be done at this point is to give our data some labels and formats, assuming these are not already on the file, since we will use them later:

```
PROC DATASETS DDNAME = <Source-library> NOLIST;
   MODIFY <Source-File> ;
   ATTRIB  INCURRED  Format=Comma9. Label = "Incurred Losses";
   ATTRIB  PAIDLOSS  Format=Comma9. Label = "Paid Indemnity Loss";
        <..etc..>;
RUN;
```

Step 2 : Transformation of the Data to Cumulative Developments; This is (or can be) a direct application of "row arithmetic". It involves the same three steps. The first is a rotation so that accident Periods are rows and Development Periods are columns.

```
PROC TRANSPOSE  DATA = <sourcedata>
                  OUT = WORKFIL1
               PREFIX = DYR;
                 NAME = ORIGVAR
                LABEL = VARLABEL
    ID        DEV_YR;
    BY        <sortbreak criteria> ACC_YR;
    VAR       INCURRED PAIDLOSS PAIDALE <etc, etc>;
    IDLABEL   DEV_YR;
RUN;
```

The result is shown in Exhibit 2.0-A below. (Compare this to Exhibit 1.0 above.)

Exbt 2.0-A After First Transpose -- Decumulated Triangle

		←Development Year→						
ITEM	ACC YR	1	2	3	4	5	6	--- Notes ---
INCURRED	1993	23,456	7,765	2,563	1,117	330	123	The ID variable is
INCURRED	1994	26,781	9,090	2,241	1,229	1,211	.	Development Year,
INCURRED	1995	29,637	8,265	2,625	1,434	.	.	DEV-YR.
INCURRED	1996	31,567	7,444	4,133	.	.	.	The VAR list consists
INCURRED	1997	36,442	4,870	.	.	.	.	of variates: e.g.
INCURRED	1998	35,113	.	.	.	.	,	INCURRED, PAIDLOSS

Next the data is now rolled forward to form the cumulative developments, using the DATA-step below. Note the upper bound **DEVELTO** of the DO-Loop. The final reporting on each accident year varies with the age of the year. For 2003, only Development year 1 is available as of 12/31/2003. For 2000, four development years are available.

```
DATA WORKFIL2;
  ARRAY DEVLOPMT(*) DYR1-DYR6;
  FORMAT  DYR1-DYR6  COMMA9.
  SET WORKFIL1;
  DEVELTO = 2003 – ACC_YR + 1
  DO JJ = 1 TO DEVELTO;
     If DEVLOPMT(JJ) = . THEN DEVLOPMT(JJ) = 0;
     If JJ GE 2 THEN DEVELOPMT(JJ) = DEVLOPMT(JJ) + DEVLOPMT(JJ-1)
     END;
RUN;
```

Exbt 2.0-B After Cumulation of the Triangle Data

		←Development Year- →						
ITEM	ACC YR	1	2	3	4	5	6	--- Notes ---
INCURRED	1993	23,456	31,221	33,784	34,901	35,231	35,454	This is done using a
INCURRED	1994	26,781	35,871	38,112	39,341	40,552	.	DATA step and an ARRAY
INCURRED	1995	29,637	37,902	40,587	42,021	.	.	of the Development
INCURRED	1996	31,567	39,011	43,144	.	.	.	Amounts. A DO-Loop
INCURRED	1997	36,442	41,312	.	.	.	.	forms the summations
INCURRED	1998	35,113	.	.	.	.	.	

The final step of this “row arithmetic” is to rotate the data back to a configuration where the columns are again data items, like INCURRED (Incurred losses) and Development Year is once more a record key. The code to do that is as follows:

```
PROC TRANSPOSE DATA = WORKFIL2
                 OUT = WORKFIL3 (DROP=_NAME_)
               LABEL = DEVYRCH
    ID       ORIGVAR;
    BY       <sortbreak criteria> ACC_YR;
    VAR      DYR1-DYR6;
    IDLABEL  VARLABEL;
RUN;

DATA WORKFIL4 (DROP=DEVYRCH)
    SET TEMPFIL3;
    DEV_YR = Input(Left(DEVYRCH,2.))
RUN;
```

The nature of the data in WORKFIL4 is shown in Exhibit 2.0-C below – compare with Exhibit 1.0:

Exbt 2.0-C Loss Data Cumulated, Rotated Back:

ACC-YR	DEV-YR	INCURRED	PAIDLOSS	…etc.	-- Notes --
::	::	::	::		
2000	1	29,637	::	::	Each Development
2000	2	37,902	::	::	Period is the sum
2000	3	40,587	::	::	sum of the changes
2000	4	42,021	::	::	in that period and
2001	1	31,567	::	::	and the ones in all
::	::	::	::	::	previous periods.

In the above code, you will note we used a follow-on DATA-step to convert the character version of Development Year back to a numeric version. You may recall we also did that in the "double transpose" of the Sales data (section 3.0).

After glancing back at the triangle we want to create and the rotation just executed, an astute reader might ask why we could not instead have simply used a different Proc Transpose execution and arrived at the desired triangle display directly, using as source the dataset created after developments were summarized.. The following code would have done that:

```
PROC TRANSPOSE DATA = FINALTRI
                 OUT = WORKFILE2 (DROP=_NAME_) /* File in Exhibit 2.0-B */
              LABEL = DEVYRCH
    ID        ACC_YR;
    BY        <sortbreak criteria> ORIGVAR VARLABEL;
    VAR       DYR1-CYR6;
    IDLABEL   ACC_YRL;
RUN;
```

and the result would be the final triangle shown earlier. There is a reason for NOT proceeding directly in this way: the desire to form derived statistics from the data. Two of the examples of derived statistics given earlier – Percent Disposed Claims and Average Reported Claim Value – are ratios that can only be formed AFTER the data is rolled up. Hence we have the rotation back to the form where the variates are SAS Variables. This leads us to Step 3.0.

Step 3.0 Calculate Derived Statistics and form Final Triangles: Here we address the need of end-users for triangles of derived quantities, such as "average reported claim value". As noted in the above paragraph, any such calculation involving products or quotients must be calculated using the cumulated (as-of) triangle data, since forming the ratios or products from change data (as it appears on the source file) and then cumulating the result leads to nonsense. The final "row arithmetic" output of cumulated amounts, WORKFIL4, is exactly what we need as the spot to add these additional statistics (variates). The following data step will add the four derived variates noted earlier.

```
DATA WORKFIL5 (DROP=DEVYRCH)
    Attrib OSRESERV  Format=Comma9. Label="Outstanding Reserve";
    Attrib NETCREAT  Format=Comma9. Label="Net-Created Claim-Ct";
    Attrib PCTDISPD  Format=Percent7.1 Comma9. Label="Percent Disposed";
    Attrib AVEINCRD  Format=Comma9. Label="Outstanding Reserve";
    SET TEMPFIL4;
    OSRESERV = INCURRED - PAIDLOSS;
    NETCREAT = CREATCNT - SETLDCNT;
    PCTDISPD = (SETLDCNT + CANCLCNT) / CREATCNT;
    AVEINCRD = INCURRED / CREATCNT;
RUN;
```

This augmented file can be rotated into the final form of the triangle. And at this point we note the recurrence of an interesting issue we encountered earlier: one of the new variates is a percentage rather than an aggregate that can be displayed as a comma edited number. Remember the "throw in some crap" routine? If we want the PROC PRINT or PROC REPORT output of the triangle to look pretty – with the dollar amounts and claim counts comma edited and the percent disposed ratio expressed as a percentage – then this is the time to add to the DATA step above the little statement

```
RETAIN CRAP "Junk";
```

And the rotation to final report form will now create entries that are character strings, with the numeric contents in proper display format. The dollars will appear as they are shown in the original Incurred loss triangle exhibit and the percent disposed amounts will appear automatically as percentages in that latter triangle display. Now we transpose the data:

```
PROC TRANSPOSE DATA = PRINTFIL
                 OUT = WORKFILE5 (DROP=_NAME_)
              PREFIX = AYR
                NAME = ORIGVAR
               LABEL = VARLABEL
    ID        ACC-YR;        /* Accident Year */
    BY        <Other Sort-break Criteria> DEV-YR;
    VAR       INCURRED  PAIDLOSS  PAIDALE   OSRESERV  AVEINCRD
              CREATCNT  SETLDCNT  CANCLCNT  NTCRTCNT  PCTDISPD
    IDLABEL   ACC_YR;
RUN;
```

CRAP;

The output file will now have the variates as record keys and the columns (variables) will be **AYR1998-AYR2003**. A simple PROC PRINT or PROC REPORT can create the triangle display as per the code following. Note how the SAS Label of the original variate is picked up as part of the title statement.

```
TITLE2 "Data Item being run #byval(VARLABEL)";
TITLE3 "(Dollars in Thousands)";
```

```
OPTIONS NOBYLINE;

PROC PRINT DATA = PRINTFIL LABEL;
    BY  <Other Sort-break Criteria>  ORIGVAR VARLABLE;
    ID  DEV_YR;
    BY       <Other Sort-break Criteria> DEV-YR;
    VAR  AYR1998-AYR2003
    WHERE  ORIGVAR NE 'CRAP';
RUN;
```

And, this is the kind of report that the code will produce, consistent with the earlier loss triangle depiction:

Accident Year Triangle-Report
Data Item Being Run: Incurred Losses
(Dollars in Thousands)

DEV-YR	1998	1999	2000	2001	2002	2003
1	23,456	26,781	29,637	31,567	36,442	35,113
2	31,221	35,871	37,902	39,011	41,312	.
3	33,784	38,112	40,587	43,144	.	.
4	34,901	39,341	42,021	.	.	.
5	35,231	40,552	.	.	.	.
6	35,454	.	.	.	.	.

Processing Efficiency, Again : On the way to the final triangle output, the data goes through several processing steps – DATA steps and PROC TRANSPOSE executions, each of which is rather simple. Is this processing and reprocessing of the loss data a matter of concern? The answer hinges on when you make the triangles.

In the actual SAS RESMENU application in Reserving Systems at The Hartford, the source triangle data is stored in the depicted "Source" format of calendar changes. The cumulative triangles are NOT formed as part of production updates to the data base. Maintenance of the data in this change form is done for both storage space reasons and for ease of production update of the data repositories. When a user triggers a report request, the data extracted as calendar changes, and only at that point is the data put through the kind of transformation sequence shown above.

Answering the question then harks back to the earlier discussion of processing efficiency versus code design considerations, at the end of section 6.0 on "row arithmetic". Because, relative to the source data, the extracted data is very small, the processing time of the mulit-step-sequence is also quite quick. And since the processing time is quick, there is no processing advantage to be gained by using the alternative "Big DATA-step" (and array manipulation) solution to delivering the final triangle data. So the choice of method hinges instead on the relative maintainability of the code, and the set up of the code in rather small bite-sized pieces, with the generic capabilities of PROC TRANSPOSE, offer a far easier application to maintain and adapt to a generic macro. This was a key appeal for us when we set up the triangle reporting software at The Hartford.

8.0 SOME CONCLUDING REMARKS:

As we noted at the outset PROC TRANSPOSE is a powerful – and (I again emphasise) somewhat underutilized -- data manipulation tool in Base SAS. It's most obvious employment is clearly making landscape reports from normalized data, or (alternatively) changing someone's landscape data file into something more easily manipulated by SAS. Basically, it switches the significance of row and column identifiers, either globally or selectively within BY-groups. All examples which were shown in this paper happened to use the BY-processing option.

Like so much of SAS, PROC TRANSPOSE offers more than simple direct applications, and a major purpose of this paper is to lead the reader into thinking about the tool as a component in solving more sophisticated application problems. To that end, we took some pains to walk through the Insurance Loss Triangle example at the end of the paper. Given something to do to data, a question should be: in what form would it be nice to have the data in to make the task the easiest. Very often the movement of the data from its delivered form to that more desirable form is (or can be) an application of PROC TRANSPOSE. The procedure is just another very useful SAS Tool.

Have fun with it!!

Acknowledgements and References:

SAS is a Registered Trademark of the SAS Institute, Inc. of Cary, North Carolina.

PROC TRANSPOSE is discussed in the SAS PROCEDURES manual, published by the SAS Institute.

Contact Information: Ralph Leighton, The Hartford.
Phone: 860-547-3014 - E-mail: rleighton @thehartford.com