

SOME USES (AND HANDY ABUSES) OF PROC TRANSPOSE

Ralph W. Leighton, The Hartford

1.0 OVERVIEW:

PROC TRANSPOSE is a powerful data manipulation tool in Base SAS®. Using it, one can, either globally or selectively within BY-groups, switch the significance of row and column identifiers. That is to say, it permits one to switch SAS Variables with logical record keys. This capability is handy, if (for example) you use financial data which 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.)

The major part of this paper discusses a few uses (of PROC TRANSPOSE) which may seem outside of the basic purpose of the procedure as noted above. Specifically, the paper will cover the following topics:

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

Although seemingly forming a disparate set of considerations, 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 point to a generic processing solution to a data manipulation problem in Casualty Insurance Claim (Loss) data, so-called loss development triangles. The latter is discussed in Section 7.0.

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 at the right. (The general nature of this data is illustrated in the PROC PRINT "Report 1.0", printed in part on the next page.)

The following code uses PROC TRANSPOSE to rotate the display so that the data items become rows and the calendar years

become columns.

```
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;
```

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. The BY variables will be on the output SAS Dataset.

VAR-statement: This identifies the data items (variables) selected for rotation. Variables not listed in the BY statement or this list are dropped. (In the present example, TOPMAN will be dropped.)

ID-Statement: The ID statement identifies a variable whose values will supply the SAS names for vari-

===== 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

ACME ASSORTED SUPPLIES CORPORATION
REPORT 1.0
INCOME SUMMARY [File SALEDAT1]

1

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

ables on the rotated data **PRTFILE**. The variable designated **CALYEAR** supplies calendar years, a four digit number. See 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 the numeric year with "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 variable which will supply SAS Labels to the new variables on the output data, in this case **CYR1996**, etc. A natural label for a report is the year value itself, and hence the choice:

IDLABEL=CALYEAR.

LABEL= option: The LABEL= option designates a SAS Variable (character) 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 does, and the SAS Labels will be saved in a variable called **ITEMNAME**.

NAME= option: Similarly, the NAME= option designates a user-selected variable name for a character variable on the output file which will hold

original SAS Variable Name. For reference here (and use later), we store the names in character variable **ORIGVAR**.

The PROC PRINT of the output SAS Dataset **MYFIRM.ROTATED1** created by PROC TRANSPOSE produces Report 2.0 (top of next page). The original variables all had COMMA SAS Formats -- note how the rotations preserved the COMMA formats.

Note also the missing values for Connecticut Auto Parts in 1998 and New York Auto parts in 1996. There were no source records for these combinations on the source file **SALEDAT1**. Thus, when the SAS dataset was rotated, a missing entry was created.

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;
```

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	
E T C .							

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

A fragment of the report appears at the bottom of the page. The ID statement references the **ORIGVAR** variable on **ROTATED1** to pick up the names of the original numeric variables. The IDLABEL statement references **ITEMNAME** to pick up the original SAS Labels.

Note the missing values entry for Connecticut Auto Parts in 1998. The double rotation has had the effect of "filling in" (with null records) entries for missing combinations on the source report. 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.

Note that, to recover **CALYEAR** as a numeric variable, I had to add a DATA-step. The values of **CALYEAR**, remember, were SAS Labels for the indexed list in the transposed data set **ROTATED1**. So using the LABEL= option on the rotation back, we can recover the 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**.

You will also note that **TOPMAN** (the sales person name) does not happen to be on this report. It was not one of the variables we transposed on the first rotation (i.e. 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 Business	State	Calen Year	Sales Revenue	Profit fr-Sales	State Sales-Tax	Salesman Commission	Other 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 of SALEDAT1:

```
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 = .;
```

Suppose that I rotated only these three new elements (i.e. as the VAR list in a PROC TRANSPOSE). Then a PROC PRINT of the result would show a nice rotated display similar to Report 2.0, with the PERCENT formats preserved, just as the COMMA formats were in

the initial example using the dollar items.

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;
```

A PROC PRINT of the resulting **ROTATED3** is shown below 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. But a SAS variable cannot have different 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.

Clearly one can get around the formatting issue by making a DATA-step report instead of PROC PRINT. There is nothing wrong with this solution path. However, a DATA-step report will involve some procedural code to space out the data nicely in columns and a

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 .						

“header” subroutine to supply the column headers with each page break. If you have to make a lot of these reports, this kind of coding can get tedious.

An alternative solution path 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  COMMISSNS
    COMRATIO   PROFIT    STAXES
    STXRATIO   COSTS    TOPMAN;
IDLABEL CALYEAR;
RUN;
```

If you PROC PRINT this file, you will get the Report 5.0 shown on this 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.

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.

What then did SAS do to the values of the original numeric variables? 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.

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 report like the one below, but don’t happen to have a character variable to stuff into the VAR list. Simply create an intermediate file **TEMPFILE** with a

ACME ASSORTED SUPPLIES CORPORATION							1
Report 5.0							
INCOME AND RATIO SUMMARY - DATA ACROSS YEARS [File NICEFILE]							
Line-of Business	State	Original Variable	Income Item Descrip	1996	1997	1998	
Auto-Parts	CT	SALES	Sales Revenue	329,746	281,288		
		PCTCHGSL	Pct-Chg in-Sales	.	(14.7%)		
		COMMISSNS	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%	
		COMMISSNS	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%	
		COMMISSNS	Salesman Commission		101,324	110,553	

ACME ASSORTED SUPPLIES CORPORATION
Report 5.0

1

INCOME AND RATIO SUMMARY - DATA ACROSS YEARS - [file NICEFILE] - Cleaned Up

Line-of Business	State	Original 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...
		COMMISSNS	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 .

character variable, like **CRAP** (below):

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

Now transpose this temporary file.

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

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 remember to remove it at report time.

This section concludes with three observations regarding the transposing of data. First, 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 was clearly taken advantage of in the examples so far discussed. Their absence would have meant more code.

Secondly, in the current example (Report 4.0) there were a couple of subtle changes in formatting from that on Report 2.0. 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 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 left justified over the columns, since the **CYR** variables are character.

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, however, requires pre-processing the output of PROC TRANSPOSE through a second DATA step before PROC PRINT-ing it, as for example:

```
DATA PRNTFILE
  ARRAY ITEMS(*) $9 CYR1996-CYR1998;
  SET MYFIRM.NICEFILE;
  DO JJ=1 TO DIM(ITEMS);
    IF ITEM(S)JJ) = ` `
    THEN ITEMS(JJ) = "...N-A...";
  END;
```

Note the specification "\$9" in the ARRAY statement to preserve the length of the **CYR**'s: If you don't specify a length, the strings will contract to \$8 and the dollar amounts on the reports will lose the last digit.

The box at the top of this page shows the effect of these two cosmetic clean-up steps.

The third and final observation has to do with the target use of transposed mixed format data. What went into Report 4.0 looks 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 3.0 (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:

If you have an indexed list of, say, numeric variables, like:

```
ACMO1 - ACMO256
```

and you want to assign SAS Labels to them, you have some choices:

- You can write out 256 LABEL statements in your SAS code (“The Method of Exhaustion”).
- You can use a macro solution. One way is to code a little macro that will generate 256 LABEL statements and then invoke it in the SAS Code. 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;
```

- You can use PROC TRANSPOSE.

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;

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

DATA DUMMY03;
  SET DUMMY02;
  STOP;
```

Note that this last file **DUMMY03** has no observations. Now use a DATA step to append **DUMMY03** to the data set needing the SAS Labels for the indexed list **ACMO1-ACMO256**:

```
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 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 created once and then used multiple times.

The use of the DATA step above to bring in the labels from **DUMMY03** means the dataset <<sourcefile>> must be read through. If you try to use PROC APPEND to add the null file **DUMMY03** to the end of the data <<sourcefile>>, the SAS Labels will not be picked up. On the other hand, in the SET statement in a DATA step, the order of appearance of the two files <<sourcefile>> and **DUMMY03** doesn't matter. Since presumably <<sourcefile>> does not have SAS labels 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 author advocated using DATA step solutions to such problems. Depending on the situation, such a solution can be appropriate. 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("COMMISSNS")
    HLDCOMMS(JJ)=ITEMS(JJ);
  WHEN("STAXES")
    HLDCOMMS(JJ)=ITEMS(JJ);
  OTHERWISE;
  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.
- ◆ 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 since there are only two such items.

An alternative approach to form **ROTATED3** is the double-transpose ("flip-flop"). 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. Then a DATA step calculates the derived items in the same manner as they were calculated in Section 4.0 -- i.e. using simple arithmetic (and no DO loops). Finally a second transposition restores the data to its original configuration, as **ROTATED3**. The code 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
                ID    ORIGVAR;
                BY    PRODUCT RSTATE;
                VAR   CYR1996-CYR1998;
                IDLABEL ITEMNAME;
RUN;
*****;
DATA WORKFIL2;
  ATTRIB  COMRATIO  FORMAT=PERCENT7.1
          LABEL="Commissn Ratio";
  ATTRIB  STXRATIO  FORMAT=PERCENT7.1
          LABEL="State Tax-Pct";
  ATTRIB  SALESCHG  FORMAT=PERCENT7.1
          LABEL="Pct-Chg in-Sales";
  SET WORKFILL1;
  CALYEAR = CYRCH;
  COMRATIO = COMMISSNS/(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   COMRATIO PCTPROFT
      STXRATIO CTRRATIO;
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.)

It is worth comparing the two approaches.

- ◆ **MAINTENANCE:** The DATA-step has a fair number of indices and loops; the program shown is 47 statements long. If you need to add more calculated items -- e.g. a Percent Profit (or Profit Ratio) -- this code will expand as additional loops and temporary storage arrangements (arrays) are set up. Now a particular application might allow some code reduction by using two- or three- dimensional arrays. While this kind of array usage might abbreviate code volume, it can also lead to headaches by obfuscating what is getting calculated.

The TRANSPOSE approach seems to be easily adapted to enhancements. If one adds more source items (to **ROTATED1**) and 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.

- ◆ **EFFICIENCY:** 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 an consideration for any application involving very large record volumes. But, for an application involving relatively small files, the number of passes should be a non-issue.

7.0 “FLIP-FLOP-FLAP” IN CASUALTY LOSS DEVELOPMENT TRIANGLE ANALYSIS

The double transpose approach is used in an actuarial data retrieval application at The Hartford, the topic of this last section of the paper.

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). It is thus no surprise that the biggest liability in such a business is the reserve for un-paid claims, both those reported (and as yet unpaid) and those yet to be reported on past business written. There is always uncertainty as to exactly how large this reserve is, and one of the actuary’s functions is to evaluate claim patterns as a test for the adequacy (or redundancy) of the posted reserves. One important element in such analyses is the projection of loss development triangles, a mini-example of which appears as the fourth display in the exhibit on the next page.

Such a report shows, by the accident period of the losses, the (usually) upward progression, by valuation date, of claim statistics such as:

- incurred losses (claim amounts reported to the insurance company);
- created claim counts (numbers of claims reported),
- paid losses (moneys paid out to claimants); or
- paid allocated expenses (moneys paid out in the handling of claims).

The source data for displays 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 (in summary form) is shown suggestively in the first display in the same exhibit on the next page. Thus the delivery of data to the end-user (the actuary) involves three steps:

- ◆ Calculation of Development Period based on the accident period and the reporting date. This is shown as column “DEV-YR” on the top display of the source data in the figure.
- ◆ Transformation of the Form of the Data; This first involves rotation and the formation of cumulative developments from the change amounts. Now there may be not change amounts, here and there, for later development periods. These need to be filled in, hence the intermediate form of the data shown as the first rotation. Missing values are converted to zeros. The data is then cumulated (rolled forward) by development period, the result shown on the third display in the figure on the next page. The already alluded-to report form is shown as the fourth and last display on the figure. In this last rotation, missing accident years for lines with incomplete history are filled in.
- ◆ Triangles of Derived Statistics: At user request, one might need to calculate triangles of derived quantities, such as “average reported claim value”. Any such calculation involving products or quo-

1.0 Source Data:						
ACC-YR	REPT-YR	DEV-YR	INCURRED	PAIDLOSS	...etc.	
::	::	::	::	::	::	
1995	1995	1	29637		::	These
1995	1996	2	8265		::	are the
1995	1997	3	2685		::	annual
1995	1998	4	1434		::	summaries
1996	1996	1	31567		::	of the
1996	1997	2	7444		::	transations
::	::	::			::	

2.0 First Transpose -- Decumulated Triangle						
←-----Development Year-----→						
ITEM	ACC YR	1	2	3	4	5
INCURRED	1993	23456	7765	2563	1117	330
INCURRED	1994	26781	9090	2241	1229	1211
INCURRED	1995	29637	8265	2625	1434	.
INCURRED	1996	31567	7444	4133	.	.
INCURRED	1997	36442	4870	.	.	.
INCURRED	1998	35113

3.0 Cumulation of the Triangle Data						
←-----Development Year-----→						
ITEM	ACC YR	1	2	3	4	5
INCURRED	1993	23456	31221	33784	34901	35231
INCURRED	1994	26781	35871	38112	39341	40552
INCURRED	1995	29637	37902	40587	42021	.
INCURRED	1996	31567	39011	43144	.	.
INCURRED	1997	36442	41312	.	.	.
INCURRED	1998	35113

4.0 Second Transpose -- Triangle-Report Layout						
←-----Accident Year-----→						
ITEM	DEV-YR	1993	1994	1995	1996	1997
INCURRED	1	23456	26781	29637	31567	36442
INCURRED	2	31221	35871	37902	39011	41312
INCURRED	3	33784	38112	40587	43144	.
INCURRED	4	34901	39341	42021	.	.
INCURRED	5	35231	40552	.	.	.
INCURRED	6	35454

tients must be calculated using the cumulated triangle data. They make no sense if calculated from source file change data. For such computed items, the method used by Reserving Systems at The Hartford rotates the data so that the original variates (items like Incurred losses) are again SAS Variables. The calculation is performed and the data is then rotated back to the report form. This is nothing more than the "flip-flop" row-arithmetic play described in Section 6.0.

In the actual SAS application in Reserving Systems at The Hartford, the source triangle data is stored in the "Source" format above, for space reasons. These files constitute a good size "data mart", because they maintain history at many levels of line structure (and other considerations) and because the data is monthly (not yearly), with a scope of well over twenty years,

accident date by reporting date. As actuarial users request triangles interactively through SAS-AF, the application first extracts the data needed and then performs on the fly the triangle accumulation and the requested special item computations. Since the data volumes processed by this reporting application are relatively small, they lend themselves to the kind PROC TRANSPOSE solutions this paper describes.

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.