Paper 2774-2018

Table Look-Up Techniques: Is the FORMAT Procedure the Best Tool for the Job?

Andrew T. Kuligowski, HSN
Swati Agarwal, Optum

ABSTRACT

SAS® programmers have employed user-written formats via PROC FORMAT to perform table look-ups for as long as PROC FORMAT has been available. There is no question that it is a viable technique – but is it the best way to attack the problem?

This paper and associated presentation will look at how PROC FORMAT can be used to facilitate table look-ups. User generated Formats can be employed to assign descriptive labels to data values, create new variables, and identify unexpected values. PROC FORMAT can also be used to generate data extracts and to merge data sets. The computer resources necessary to execute this technique will be examined, and contrasted with alternate approaches such as the DATA Step MERGE statement and SQL JOIN.

KEYWORDS

Base SAS, PROC FORMAT, MERGE, JOIN

THE BASICS OF PROC FORMAT

WHAT IS PROC FORMAT?

The FORMAT procedure enables you to define your own informats and formats, when the numerous standard set of informats and formats that comes with SAS proves insufficient for your needs. These can be hardcoded within your PROC FORMAT, or generated dynamically from within a dataset. In addition, you can print the parts of a catalog that contain informats or formats, store descriptions of informats or formats in a SAS data set.

DIFFERENCE BETWEEN FORMAT AND INFORMAT?

Informats determine how raw data values are read and stored. **Formats** determine how variable values are printed. In other words, the **informat** converts (incoming) and the **format** prints (outgoing).

With **informats**, you can do the following:

- Convert a number to a character string (for example, convert **1** to **"YES"**).
- Convert a character string to a different character string (for example, convert **"YES"** to **"OUI"**).
- Convert a character string to a number (for example, convert **"YES"** to **1**).
- Convert a number to another number (for example, convert **0** through **9** to **1**, **10** through **100** to **2**, and so on).

Formats allow you to do the following:

- Print numeric values as character values (for example, print **1** as **"MALE"** and 2 as **"FEMALE"**).
- Print one character string as a different character string (for example, print **"YES"** as **"OUI"**).

- Print numeric values as a "picture", using a template (for example, print **9458763450** as **"945-876-3450"**).

USER DEFINED FORMAT

SAS supplies tons of formats to display – number, characters, strings, dates etc. For example:

- The **$CHAR.** format will display a character string, preserving all leading and trailing spaces.
- The **COMMA.** format will display a number with comma punctuation.
- The **DATE.** format will display a SAS® serial date in familiar day-month-year (ddMONyy) notation.

Despite the many built-in formats available in SAS, you can easily run into a situation where a format does not exist to suit your needs. Your data may contain a column called "sex" (or "gender", if you prefer) that has values of 1 and 2 representing females and males respectively. No SAS supplied format will de-reference these coded values for you. You could use a data step to create a new column to de-reference these coded values.

For example, moving to a very slightly more complicated example:

```
data new;
   set old;
   length grade $ 6;
   if      score = A then grade = 'Pass';
   else if score = B then grade = 'Pass';
   else if score = C then grade = 'Pass';
   else if score = D then grade = 'Pass';
   else if score = E then grade = 'Fail';
   else if score = F then grade = 'Fail';
run;
```

This approach does not cause any issues with small datasets. If, however, you have a dataset with millions of rows, then you will be wasting the computer resources required to execute an additional DATA Step, when compared to the capability of SAS's option to employ a user defined FORMAT.

Building a user defined FORMAT can be viewed as a table lookup where VALUES are mapped to LABELS. Let us look at some table lookup examples.

Typical lookup tables use 1-to-1 or many-to-1 mappings. As an example of a 1-to-1 table lookup, we have a data set of 1 character grading codes. When we generate reports of pass and fail, we wish to map grading code values into literal labels. The 1-to-1 mapping is illustrated here:

```
'A' = 'Pass'
. . .
'F' = 'Fail'
```

If we have many grading codes, they can be mapped, assigned, or grouped to the appropriate label in a many-to-1 mapping:

```
'A', 'B', 'C', 'D' = 'Pass'
. . .
'E', 'F' = 'Fail'
```

Writing the code to accomplish the above is pretty simple:

```
proc format;
   value grade
   A = 'Pass'
   B = 'Pass'
   C = 'Pass'
   D = 'Pass'
   E = 'Fail'
   F = 'Fail'
;
run;

proc freq data = old;
   format grade grade.;
   table grade;
run;
```

The PROC FREQ following the PROC FORMAT will summarize the data by the FORMATTED value of the variable GRADE, not the original value. This is one of the more powerful uses of formats, and is only strengthened by the ability to create your own formats over and above the standard ones that come bundled in Base SAS.

USING PROC FORMAT TO PERFORM TABLE LOOKUP TECHNIQUES

In order to use user-written formats as a look-up table, it will be necessary to convert your data into formats using PROC FORMAT. This will be illustrated by taking selected fields from the SASHELP.ZIPCODE file that comes as a part of BASE SAS:

```
data  State_Lookup;
  set  sashelp.zipcode(KEEP=City State  StateCode StateName
                           ZIP  County CountyNM  TimeZone  DST);
run;
```

PROC FORMAT (Using CNTLIN=)

We will employ Character formats for our table look-ups. In order to do this, we need to structure our dataset in the format (no pun intended) that PROC FORMAT requires. There are several variables that can be used to define a given format; the limited few that we will need are shown below :

Fmtname	the name of the format to be created.
HLo	specifies a particular piece of information about a format's range, as denoted by its value. On our example, "O" stands for "OTHER", and represents the value to be substituted when a lookup value cannot be found in the table.
Label	the value to be associated with the output format.
Type	"c" for character, the value to be used in our examples. (n=numeric format, p = picture format, j = character informat, i=numeric informat

PROC FORMAT – STORING / RETURNING A SINGLE RESULT

The two examples in the following table are very similar. The first is designed to perform a 1 to 1 lookup, with a key field in our lookup table being the key field in the file we are to match against. In this case, the example uses ZIP CODE (called **ZIP** in the file) to look up Time Zone (conveniently, also called **TIMEZONE**).

The second illustrates an example in which a single key field is insufficient to define a unique key in our file, County Code (called **COUNTY** in the file) to County Name (called **COUNTYNM**) In this case, we will be concatenating two separate fields to make our look-up functional; State Code (**STATE**) and County Code (**COUNTY** as previously mentioned)

Very little pre-processing is required for the initial Format. The table contains one record per ZIP Code – our key variable for this lookup. We DO make sure we use a 5 character ZIP Code, to ensure that leading zeroes are respected. The second format is not much more complicated; we do zero-pad both State Code and County Code prior to concatenating them together as a single key variable, since for example,111 could be State 1 County 11 OR State 11 County 1 without the clarification of zero-padding.

Both examples simply RENAME the variable being used as the result of the lookup to Label to accommodate the needs of **PROC FORMAT**; it is not necessary to perform any additional processing against it. It IS often useful to have a default value to use when the look-up is unsuccessful; setting the **HLO** variable to **"O"** (for "Other") lets **PROC FORMAT** know that we are providing the "what to use when **PROC FORMAT** can't find the result value.

Note that it is highly suggested to sort the values; there IS an unsorted option, **HLO="S"**, but clarity and efficiency lean towards including the sort. This sort is NOT optional for the second example. Since our table often has multiple records per state / county combo – remember, it is one record per ZIP code – the **NODUPKEY** option will allow **PROC SORT** to remove redundant records from the file prior to format creation and look-up Should the data contain duplicate key values during the execution of **PROC FORMAT**, SAS will produce an error message in the SASLOG similar to the following:

```
ERROR: This range is repeated, or values overlap: 01001-01001.
```

Finally, we invoke **PROC FORMAT**, using the **CNTLIN=** option to advise SAS that we are passing in a properly formatted dataset containing the format to be processed. For the record, we COULD concatenate our datasets together and only invoke **PROC FORMAT** the one time – after all, the **FmtName** variable clearly shows which format is being created in each observation – but we've left them separate for clarity's sake.

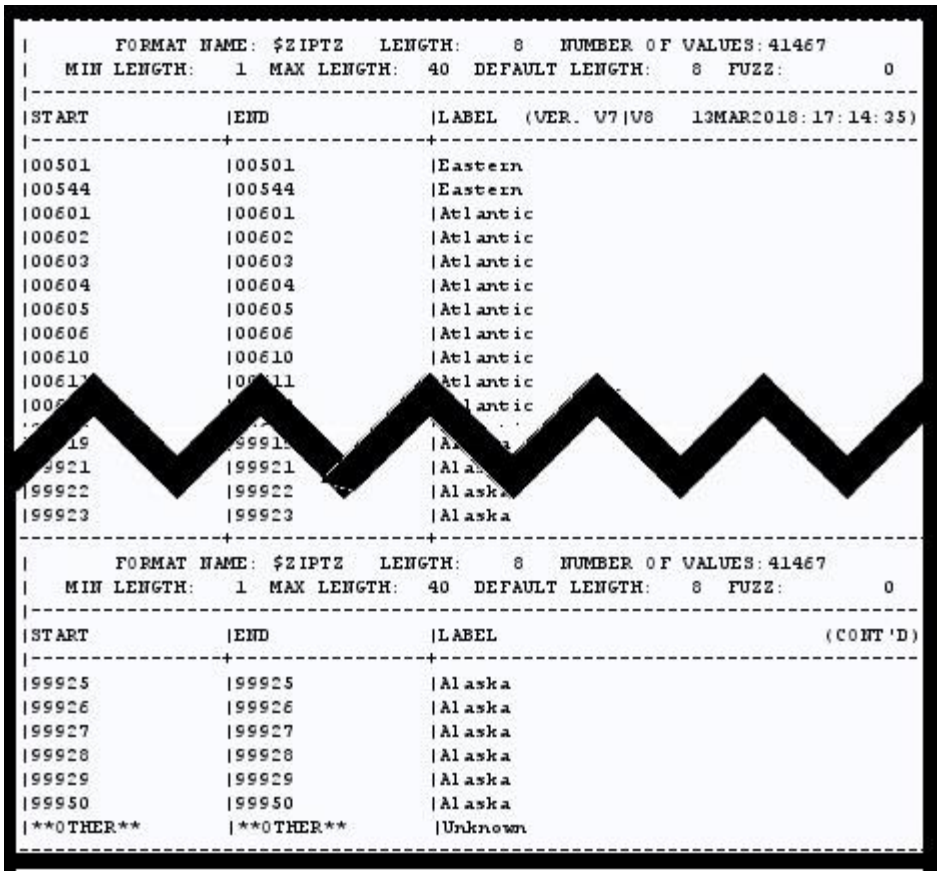
```
DATA SingleVar
   (KEEP=fmtname type start
         label   HLO);
  LENGTH fmtname $ 8
         label   $ 9
         type    $ 1 ;
  RETAIN fmtname 'ZIPTZ'
         type    'c' ;
  SET  State_Lookup
      (Rename=(TimeZone=Label))
       END=LastRec ;
  Start = PUT( ZIP, Z5. );

  OUTPUT;
  IF  LastRec  THEN  DO;
    HLO = "O";
    ***start = "**OTHER**";
    label = "Unknown";
    OUTPUT;
  END;
RUN;
PROC SORT  DATA=SingleVar;

  BY  start;
RUN;
PROC FORMAT CNTLIN=SingleVar;
RUN;
```

```
DATA MultVar1;


  LENGTH fmtname $ 8

         type    $ 1 ;
  RETAIN fmtname 'StCnty'
         type    'c' ;
  SET  State_Lookup
      (Rename=(CountyNm=Label))
       END=LastRec ;
  Start = PUT( State, Z2. ) ||
          PUT( County, Z3. ) ;
  OUTPUT;
  IF  LastRec  THEN  DO;
    HLO = "O";
    start = "**OTHER**";
    label = "Unknown";
    OUTPUT;
  END;
RUN;
PROC SORT  DATA=MultVar1
           NODUPKEY;
  BY  start;
RUN;
PROC FORMAT CNTLIN=MultVar1;
RUN;
```

As an aside, it is easily possible to verify the contents of a format that was manually created, simply by issuing the command below (using your own format, or list of formats as it is possible to specify multiple formats). :

```
PROC FORMAT FMTLIB LIB=Work.Formats;
  SELECT $Stcnty;
RUN;
```

It is also possible to examine the Formats library from the EXPLORER window, and double-click on the format to be displayed. This triggers the command above to be generated and issued behind-the-scenes, producing a table similar to the following (the jagged line simply indicating that we are suppressing the display of thousands of lines of ZIP Codes, of course):



FORMAT NAME: $ZIPTZ LENGTH: 8 NUMBER OF VALUES: 41467		
MIN LENGTH: 1 MAX LENGTH: 40 DEFAULT LENGTH: 8 FUZZ: 0		
START	END	LABEL (VER. V7|V8 13MAR2018:17:14:35)
00501	00501	Eastern
00544	00544	Eastern
00601	00601	Atlantic
00602	00602	Atlantic
00603	00603	Atlantic
00604	00604	Atlantic
00605	00605	Atlantic
00606	00606	Atlantic
00610	00610	Atlantic
00611	00611	Atlantic
00[illegible]	[illegible]	[illegible]antic
[illegible]19	99915	[illegible]
99921	99921	Alaska
99922	99922	Alaska
99923	99923	Alaska

FORMAT NAME: $ZIPTZ LENGTH: 8 NUMBER OF VALUES: 41467		
MIN LENGTH: 1 MAX LENGTH: 40 DEFAULT LENGTH: 8 FUZZ: 0		
START	END	LABEL (CONT'D)
99925	99925	Alaska
99926	99926	Alaska
99927	99927	Alaska
99928	99928	Alaska
99929	99929	Alaska
99950	99950	Alaska
OTHER	**OTHER**	Unknown

PROC FORMAT – STORING / RETURNING MULTIPLE RESULTS

This all works when the need is to perform a single variable to single variable match. It is also possible to perform this task using **`PROC FORMAT`** to perform a look-up when two or more fields are requested from the table look-up.

The simplest method to code and to explain is by creating a 3rd variable that is the concatenation of the variables that are to be the target of the look-up. In this way, the technique is still very much like the simple 1-1 match logic, except that the result (**`"label"`**) must contain the concatenated group of variables to be retrieved. (The complication occurs at look-up time, when it is necessary to split these variables apart. The example below is simple, in that the first variable is always two characters long; other situations may require more finesse.

```
DATA MultVar;
  LENGTH fmtname $  8
         label   $ 40
         type    $  1 ;
  RETAIN fmtname 'StCity'
         type    'c' ;
  SET State_Lookup
      END=LastRec ;
  Start = PUT( ZIP, Z5. );
  label = StateCode || City ;
  OUTPUT;
  IF  LastRec  THEN  DO;
    HLO = "O";
    ***start = "**OTHER**";
    label = "Unknown";
    OUTPUT;
   END;
RUN;
PROC SORT  DATA=MultVar;
  BY  start;
RUN;
PROC FORMAT CNTLIN=MultVar;
RUN;
```

The most complex example occurs when one is attempting to perform a match on a great number ("great" being a relative term in this instance) of variables in a single record. In this case, it is possible to use **PROC FORMAT** to point to an observation number in the look-up dataset. A subsequent **DATA** step using a **SET** statement with direct access (also known as random access) will be employed to pull the correct observation for table look-up.

```
DATA AllVar;
  LENGTH fmtname $  8
         label   $ 12
         type    $  1 ;
  RETAIN fmtname 'ZIPTBL'
         type    'c' ;
  SET State_Lookup
      END=LastRec ;
  Start = PUT( ZIP, Z5. );
  label = COMPRESS(PUT(_N_,12.));
  OUTPUT;
  IF  LastRec  THEN  DO;
    HLO = "O";
    ***start = "**OTHER**";
    label = "-99";
    OUTPUT;
   END;
RUN;
```

```
PROC SORT  DATA=AllVar;
  BY  start;
RUN;
PROC FORMAT CNTLIN=AllVar;
RUN;
```

In early releases of SAS, the length of a value stored in PROC FORMAT was greatly limited. This trick would allow the user to use PROC FORMAT on variables that exceeded the size limitation for character variables. It can still be used in that instance today – in those days, character variables had a cap of 200 characters!

USING THE DATA STEP TO PROCESS USER-WRITTEN FORMATS

All of the examples provided to date require the following statement to be used in a subsequent DATA step to utilize the data stored in PROC FORMAT:

```
Value = PUT( variable, format. );
```

For the simple examples, which generally turn out to be the majority of cases in the real-world, this statement is sufficient in and of itself – once the proper results are provided for **variable** and **format**. In the "many to one" case, **variable** may be pre-processed or it may be handled in the PUT statement itself by building the combined result on the fly.

It will be necessary to disassemble the results when the result contains two or more concatenated value. This will be the inverse of whatever trick was employed when the format was first created.

For table look-up with direct (random) access, a **POINT=<*record number*>** on a subsequent SET statement is necessary to make the appropriate observation to observation connection, where **record number** is the value of _N_ as defined in the format creation..

TIME TRIALS

Due to space and time limitations, actual time trial results will be briefly reviewed during the oral presentation accompanying this paper, and will published at a future time.

It should be noted that in several test cases, the anticipated results will match the actual ones so closely that one may wonder why it was necessary to state and run the experiment. There were several reasons why a great variance might be expected, turn out to be the case, but still justify the execution of the experiment.

- Everyone can cite examples of popular wisdom that turned out not to be so wise – and in fact, were incorrect or even contradictory as proven by experimentation.
- The magnitude of difference was also important. One may be able to state with certainty that "Approach X is obviously better than Approach Y", but the degree to which that statement proved true may be surprising.
- In some cases, things that seem perfectly obvious to one person may be news to another. Further, a single statement may inspire a reader to pursue an entirely different line of thinking independent of the original experiment.

It is also important to note that this experiment does not factor in human effort. Some techniques can be difficult to code and test, and remain difficult to maintain after they have been rolled into a production environment. A slight improvement in thru-put that is achieved via logic that takes significant time to code and maintain may prove not to be worth the additional effort.

CONCLUSION

The goal of this short presentation is not to provide definitive answers as to "the best tool for the job". Factors such as data volume, machine capacity and current workload, and even personal preference / expertise that are unique to each situation will combine to provide different interpretations of the results. Rather, it is hoped that by providing the basics of how to perform table look up using PROC FORMAT and laying the groundwork of a test strategy, readers and attendees will gain insight as to how to best make this decision in their own job responsibilities – remembering that their OWN decision may be different as their OWN factors and circumstances evolve.

REFERENCES / RECOMMENDED READING

Patton, Nancy K. (1995). "IN & OUT of CNTL with PROC FORMAT" Proceedings *of the Twenty-Third Annual SAS Users Group International Conference*. Cary, NC: SAS Institute, Inc.
.http://www2.sas.com/proceedings/sugi23/Coders/p68.pdf

Ray, Craig (1987), "A Comparison of Table Lookup Techniques" Proceedings *of the Twelfth Annual SAS Users Group International Conference*. Cary, NC: SAS Institute, Inc.
http://www.sascommunity.org/sugi/SUGI87/Sugi-12-09%20Ray.pdf

SAS Institute, Inc. (2009), Base *SAS 9.2 Procedures Guide.* Cary, NC: SAS Institute, Inc.
http://support.sas.com/documentation/cdl/en/proc/61895/HTML/default/viewer.htm#a000063536.htm

SAS Institute, Inc. (2015), *SAS 9.4 Language Reference: Concepts, Fifth Edition.* Cary, NC: SAS Institute, Inc.
https://support.sas.com/documentation/cdl/en/lrcon/68089/HTML/default/viewer.htm#titlepage.htm

SAS Institute, Inc. (2017), Base *SAS 9.4 Procedures Guide, Seventh Edition.* Cary, NC: SAS Institute, Inc.
http://documentation.sas.com/?docsetId=proc&docsetTarget=p1xidhqypi0fnwn1if8opjpqpbmn.htm&docsetVersion=9.4&locale=en

SAS Institute, Inc. (2015), *SAS 9.4 Statements: Reference, Fourth Edition.* Cary, NC: SAS Institute, Inc.
https://support.sas.com/documentation/cdl/en/lestmtsref/68024/HTML/default/viewer.htm#titlepage.htm

ACKNOWLEDGMENTS

Special thanks go out to the individuals who believed in the concept of this paper and asked that it be fleshed out and presented. In addition, our gratitude goes out to our respective employers and co-workers, as well as our families, for support while we designed and executed the experiments which are documented in this presentation.

In addition, Andrew would like to offer a tip of the hat to veteran SAS Coders Craig Ray and Nancy Wilson (nee Patton), whose works on the subject inspired a young(er) programmer to continue to explore the topics of table look-ups, efficiencies, and PROC FORMAT. Two papers that left an indelible mark on his professional career are included in the REFERENCES section.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Andrew Kuligowski
HSN
St. Petersburg, FL
E-mail: KuligowskiConference@gmail.com

Swati Agarwal
Eden Prairie, MN
E-mail: epswati@yahoo.com