Paper RF-12-2015

Find your Way to Quick and Easy Table Lookups with FINDW

Joshua M. Horstman, Nested Loop Consulting, Indianapolis, IN

ABSTRACT

Table lookups involve setting the value of one variable based on the value of another variable. There are many ways to perform table lookups in SAS, and many papers have been written on the various techniques. This paper demonstrates a quick and easy method using the FINDW function and discusses when it makes sense to use this method and when more robust methods might be preferable.

INTRODUCTION

Table lookups are an essential part of the SAS programmer's toolkit. A table lookup occurs when the value for a variable must be determined (or "looked up") based upon the value of another variable. For example, demographic data might contain race or ethnicity codes such as "A", "B", and "W", and a table lookup must be performed to derive the corresponding labels such as "Asian", "Black", and "White".

A variety of methods are available to perform such a lookup, and much has been written about the various techniques in dozens of papers going back to the 1980s. Carpenter (2001) catalogs at least 10 methods, ranging from hard-coding using IF-THEN-ELSE statements to creating a custom format with PROC FORMAT to more esoteric methods such as indices and arrays, expanding further on some of these in a 2015 follow-up paper. Others who have tackled this complex topic recently include Croonen and Theuwissen (2002) and SAS Institute's own Stroupe and Jolley (2008). Dorfman and Vyverman (2004) focus on the use of hash objects for table lookups, while Piet (2000) suggests a method for table lookups "on the fly" using SYMPUT and SYMGET. This paper proposes yet another method of performing a table lookup, albeit one that is suited only for certain circumstances.

EXAMPLE #1: A SIMPLE TABLE LOOKUP

Consider this simple example: We have a SAS data set called PLANETS1 containing a variable called PLANET_NAME. The value of PLANET_NAME is the common name for one of the eight[1] planets in our solar system. We have one record per planet, but in order to correctly sort them for reporting, we need to add an ordering variable. This new ordering variable will be called PLANET_ORDER and sequentially number the planets in increasing order by distance from the sun (i.e. Mercury = 1, Venus = 2, etc.).

There are, of course, numerous ways to accomplish this in SAS. One way is to create a format and then use the PUT statement to apply the format as follows. (Note the use of the UPCASE function to make this a case-insensitive lookup.)

[1] As of the time of this writing, only eight planets are recognized by the International Astronomical Union.

```
proc format;
    invalue planets
        'MERCURY' = 1
        'VENUS'   = 2
        'EARTH'   = 3
        'MARS'    = 4
        'JUPITER' = 5
        'SATURN'  = 6
        'URANUS'  = 7
        'NEPTUNE' = 8
    ;
run;

data planets2;
    set planets1;
    planet_order = input(upcase(planet_name),planets.);
run;
```

Another method is to encode the lookup in a set of IF…THEN…ELSE statements like this:

```
data planets2;
    set planets1;
    if      upcase(planet_name) = 'MERCURY' then planet_order=1;
    else if upcase(planet_name) = 'VENUS'   then planet_order=2;
    else if upcase(planet_name) = 'EARTH'   then planet_order=3;
    else if upcase(planet_name) = 'MARS'    then planet_order=4;
    else if upcase(planet_name) = 'JUPITER' then planet_order=5;
    else if upcase(planet_name) = 'SATURN'  then planet_order=6;
    else if upcase(planet_name) = 'URANUS'  then planet_order=7;
    else if upcase(planet_name) = 'NEPTUNE' then planet_order=8;
run;
```

Yet another method would be to create a separate reference data set containing the corresponding values of PLANET_NAME and PLANET_ORDER for each planet. This reference dataset could then be merged into the main dataset by planet name to bring in the ordering variable. Alternatively, this reference dataset could serve as the basis for a hash object.

The advent of the FINDW function introduces another alternative. Here it is presented in its simplest form.

```
data planets2;
    set planets1;
    planet_order = findw(
        'MERCURY VENUS EARTH MARS JUPITER SATURN URANUS NEPTUNE',
        planet_name,' ','eir');
run;
```

THE FINDW FUNCTION

The FINDW function was introduced in SAS version 9.2. It functions in one of two ways. It can either return the character position of a word in a string, or it can return the number of the word in the string. In our case, we make use of the latter functionality.

The FINDW can accept up to five arguments, but we use only the following four:

1. STRING – the character string to be searched
2. WORD – the word for which to search in STRING
3. CHARACTER – a string listing the character(s) used to delimit boundaries between words in STRING.
4. MODIFIER – a string of characters which each modify the behavior of the FINDW function in specific ways

In the code presented above, we have made use of three modifiers. The "E" (or "e") modifier causes FINDW to return the word count corresponding to WORD in STRING as opposed to the default behavior of returning the character position. The "I" (or "i") modifier causes FINDW to ignore the case of letters when searching for WORD in STRING. Finally, the "R" (or "r") modifier removes any leading or trailing delimiters from WORD before performing the search. This is important in our example since the names of the planets have different lengths. Shorter planetary names will necessarily be padded with spaces, and spaces happen to be the delimiter we've specified.

EXAMPLE #2: A SLIGHTLY MORE COMPLEX TABLE LOOKUP

The method as presented above has both advantages and drawbacks. One serious limitation is that it only works in cases where the values to be looked up are a sequence of consecutive integers starting with 1. While such cases are not uncommon, they are hardly representative of the wide variety of situations in which one might wish to perform a table lookup.

Happily, this scheme can be expanded to accommodate any arbitrary set of values that are needed. To demonstrate this enhancement, consider another example: We have a data set called AIRPORTS1 which includes a variable called AIRPORT_CODE. The values of AIRPORT_CODE are the three-letter abbreviations associated with various airports. We wish to add another variable called CITY containing the name of the major city served by the airport.

As with our first example, this can be accomplished using a custom format, or using IF…THEN…ELSE logic, or in quite a few other ways. It can also be performed by using the FINDW function in conjunction with the SCAN function.

```
data airports2;
    set airports1;
    city=scan(
        'Omaha Chicago Indianapolis Cincinnati Detroit Cleveland',
        findw('OMA ORD IND CVG DTW CLE',airport_code,' ','eir'));
run;
```

THE SCAN FUNCTION

Unlike the FINDW function, the SCAN function has been a part of SAS software for a long time, dating back to at least sometime in the 1980s. It is essentially the functional inverse of FINDW with the "E" modifier. Rather than returning the number of a word in a string, it returns the word when given the number and the string.

The SCAN function accepts four arguments, but we will use only three:

1. STRING – the character string to be searched
2. COUNT – the number of the word to be selected from STRING
3. CHARACTER-LIST – a string listing the character(s) used to delimit boundaries between words in STRING.

In the code above, we pass a string containing all of our look-up values to the STRING argument of the SCAN function. The values are listed in the same order as the corresponding entries in the string passed to FINDW. The numeric value returned by the FINDW function serves as the COUNT argument for the SCAN function. In this example, we allow the CHARACTER-LIST argument to the default set of delimiters, which includes spaces.

EXAMPLE #3: USING AN ALTERNATE DELIMITER

You may have noticed that all of the cities included in the previous example conveniently have one-word names. Had we included Des Moines or Kansas City in our example, the code as written would not have worked properly since the SCAN function would treat each of those names as two separate words.

In a situation where the values to be looked up might include embedded spaces, it is necessary to use a different delimiter that will not be found in the data. Consider our final example: We have a data set called RESPONSE1 which includes a variable called RESPONSE_CODE. The values of RESPONSE_CODE are two-letter codes used to express the outcome of a patient's response to an experimental cancer drug.

We wish to decode those two-letter codes into the full designations used for reporting purposes. A given designation might be comprised of multiple words and therefore include embedded spaces. For that reason, we use the forward slash (/) as an alternative delimiter. This alternative delimiter must be embedded into the character string passed to the SCAN function. In addition, the CHARACTER-LIST argument of the SCAN function must include the forward slash character but not the space.

```
data response2;
    set response1;
    response = scan('Progressive Disease/Partial Response/
                     Complete Response/Stable Disease/Not Evaluable',
                    findw('PD PR CR SD NE',response_code,' ','eir'),
                    '/');
run;
```

ANALYSIS AND CONCLUSION

Using the FINDW function (possibly in conjunction with the SCAN function) to perform in-line table look-ups is not suitable in all situations. This practice has some significant drawbacks that should be considered prior to its use.

First, since the list of input values as well as the corresponding values to be returned must be embedded directly within the code, this approach is ideal only for short lists. Utilizing such a method for lengthy look-up tables could result in code that is unwieldy and difficult to read. Long lists are probably best placed in a separate data set.

Secondly, since the values are embedded directly in the code, this practice should not be used where the lists of values may change. In such situations, it may be preferable to take a data-driven approach in which the lookup table is built on-the-fly.

Finally, in-line table lookups make the most sense in situations where the lookup will only be performed in one place in the code. In applications calling for the same table lookup to be performed repeatedly, there is a strong case to be made for encoding the relationship between the two lists of values in a centralized way such as in a custom format or a separate data set. Encoding the same information in multiple places throughout a program can cause problems if the various instances are not kept meticulously synchronized when any changes are made.

Despite the above shortcomings, in-line table lookups using FINDW and SCAN do have some real advantages. They are quick and simple. The code is compact and easy to read. It's easy to modify if the list of values changes. It does not require the creation or use of other data sets. It's self-contained, which makes it convenient to reuse in another project.

Like any good craftsman, the savvy SAS programmer has many tools at his disposal and knows how and when to use each one. This technique is a handy one for a SAS programmer to keep in the toolbox.

REFERENCES

Carpenter, Arthur L. "Table Lookups: From IF-THEN to Key-Indexing." Proceedings of the Twenty-Sixth Annual SAS® Users Group International Conference, Paper 158-26. Cary, NC: SAS Institute Inc., 2001. http://www2.sas.com/proceedings/sugi26/p158-26.pdf

Carpenter, Arthur L. "Table Lookup Techniques: From the Basics to the Innovative." Proceedings of the SAS® Global Forum 2015 Conference, Paper 2219-2015. Cary, NC: SAS Institute Inc., 2015. http://support.sas.com/resources/papers/proceedings15/2219-2015.pdf

Croonen, Nancy, and Henri Theuwissen. "Table Lookup: Techniques Beyond the Obvious." Proceedings of the Twenty-Seventh Annual SAS® Users Group International Conference, Paper 11-27. Cary, NC: SAS Institute Inc., 2002. http://www2.sas.com/proceedings/sugi27/p011-27.pdf

Dorfman, Paul M., and Koen Vyverman. "Hash Component Objects: Dynamic Data Storage and Table Look-Up." Proceedings of the Twenty-Ninth Annual SAS® Users Group International Conference, Paper 238-29. Cary, NC: SAS Institute Inc., 2004. http://www2.sas.com/proceedings/sugi29/238-29.pdf

Piet, John M. "Table Lookup on the Fly Using SYMPUT and SYMGET." Proceedings of the Twenty-Fifth Annual SAS® Users Group International Conference, Paper 78-25. Cary, NC: SAS Institute Inc., 2000. http://www2.sas.com/proceedings/sugi25/25/cc/25p078.pdf

Stroupe, Jane, and Linda Jolley. "Using Table Lookup Techniques Efficiently." Proceedings of the SAS® Global Forum 2008 Conference, Paper 095-2008. Cary, NC: SAS Institute Inc., 2008. http://www2.sas.com/proceedings/forum2008/095-2008.pdf

CONTACT INFORMATION

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

Joshua M. Horstman
Nested Loop Consulting
8921 Nora Woods Drive
Indianapolis, Indiana 46240
317-414-9584
josh@nestedloopconsulting.com