# Table Lookup on the Fly Using SYMPUT and SYMGET

John M. Piet
IBM Storage Subsystems Division, Tucson Arizona

It is often necessary to search a large data set or file for information on a list of items, or to update a large data set with data accompanying the items in the list. The method most commonly suggested for doing this is to sort both data sets by the item and merge them. The sort/merge method is efficient when the large data set is already sorted by the search item, but more often than not, the search item is located in a non-indexed field in one or more files or data sets not under your control. Not only are the manipulations involving the large data set time consuming, but it is also very easy to mis-code the merge data step introducing insidious errors into your data. Other suggested methods involve setting up SAS formats for the search items. This is much more efficient, but again, setting up the code can be tricky.

I called this paper "table lookup on the fly", because there are many situations in which the search table can be set up using only one line of code by using a call to SYMPUT, and the data extracted by using only one line of code using a call to SYMGET. This is possible when the search item meets the requirements of a SAS macro variable. It must begin with an alphabetic character, contain only alphanumeric characters, and be the proper length. I run my SAS applications on an MVS system on an IBM mainframe, and the variable length is limited to only eight characters, however I understand the length may be 2**36 on some systems. To insure that the macro variable starts with a valid character and to avoid conflict with other macro variables, I usually concatenate an alphabetic character to the search key. This is best seen with some examples:

## EXAMPLE 1: EXTRACTING RECORDS FROM A LARGE FILE

You design various circuit cards which are identified with a seven character alphanumeric part number. You want to search a large data file which contains
customer orders and pull all orders for any of your parts. Here is the code:

```
DATA _NULL_;        /* read list of your part numbers */
 INFILE PARTLIST;
 INPUT PARTNUM $;
 CALL SYMPUT('P'||PARTNUM,'X');       /* set up search table */

 DATA  ORDERS;            /* create data set from order file  */
 INFILE ORDERFIL;
 INPUT . . PARTNUM $ . . . .;
 IF SYMGET('P'||PARTNUM)='X';       /* look up in search table */
```

The first step creates a macro variable of the form 'Pnnnnnnn' with a value of 'X', where nnnnnnn is the part number. In the second step, SYMGET will return the value 'X' if a macro variable was generated by the first step. The 'P' had to be concatenated to the part number because a SAS variable must start with a letter or underscore. This will probably be true for most uses of this method, limiting the search items to those of length seven or less. Note, only two lines of code were required to set up and search the table. Also, the size of the large data set is not a limiting factor. The only records kept in the ORDERS data set are those that were extracted. This example will run against a million records as easily as a thousand.

Let's assume that the ORDERS file has two part number fields that need to be checked. Think of the extra sorts and merges, not to mention variable renaming, that would be required to check both part fields using the sort/merge method. The 'ON THE FLY' method requires only one extra line of code in the extraction step:

```
IF SYMGET('P'||PARTONE)='X'  OR       /* check for 1st part number  */
   SYMGET('P'||PARTTWO)='X';          /* check for 2nd part number */
```

EXAMPLE 2: EXTRACTING DATA FROM A LARGE FILE

You have a list of part numbers and quantities that were ordered during the month and need to generate a report which also includes the part name and the unit cost of each part. These are contained in another large, non-sorted file. Here is the code:

```
DATA PARTS;                          /* read in failed parts */
INFILE PARTS;
INPUT DATE PN QTY;
CALL SYMPUT('C'||PN,'.');             /* set up search table */

DATA _NULL_;
INFILE MASTER;
INPUT PN COST PRTNAME;
IF SYMGET('C'||PN)^='  ';               /* look up in search table */
CALL SYMPUT('C'||PN,COST);              /* add cost to pn in table */
CALL SYMPUT('N'||PN,PRTNAME);   /* add name to pn in table */

DATA PARTS;
LENGTH PRTNAME $ 15;
SET PARTS;
COST = 0 +SYMGET('C'||PN);            /* assign cost  */
PRTNAME = SYMGET('N'||PN);   /* assign part name */
```

A second macro variable starting with an 'N' was used to pull in the part name. Notice how one line of code creates a second lookup table. SYMGET returns a character string of length 200, making it wise to define the length of the receiving variable to avoid problems. The key to increasing efficiency is the ability to use the 'DATA _NULL_' in processing the large file. This saves the time and storage required to build a large SAS data set.

EXAMPLE 3: UPDATING A LARGE DATA SET

You want to update a large data set with the latest cost of some purchased parts.

```
DATA _NULL_;              /* read in list parts and cost */
INFILE PARTS;
INPUT PN COST 6.2
CALL SYMPUT('C'||PN,COST);  /* set up table with cost in it */

DATA  NEW(DROP=NEWCOST);
SET OLD;
NEWCOST = 0 + SYMGET('C'||PN);  /* get updated cost from table */
IF NEWCOST>0  THEN COST = NEWCOST; /* cost was in table */
```

SYMGET returns a character string. A temporary variable is used to convert the table value to a numeric value. The conditional assignment assures that the data set is only updated if there was a new value in the table. More than one field in the data set could be updated by concatenating a different first character, as was done in the second example. These values could be pulled from several different source files, but only one pass is required through the data set being updated. If no macro variable exists, SYMGET returns a blank string and writes a warning message to SAS.LOG. To keep from getting many pages of warnings about illegal calls to SYMGET, use an OPTIONS ERRORS=1.

If your search key is too long for a macro variable, this technique can still be used to reduce the number of records in large data set for the sorting/merging method. Instead of using the entire key, the table can be set up with seven characters from the search key. Using the first example, the code to set up the table becomes,

```
CALL SYMPUT('P'||SUBSTR(PARTNUM,1,7),'X');
```

and the code to select the records becomes:

```
IF SYMGET('P'||SUBSTR(PARTNUM,1,7)='X';
```

This will significantly reduce the size of the intermediate data set and make for a much faster sort and merge. I recently ran a test case with 7 search keys and over 612,000 records in the big file. There were 107 observations selected from the big file. The sort/merge method took 33.7 seconds of CPU time. The ‘ON THE FLY’ method took 9.8 seconds using the full key and 11.7 seconds using a partial key.

The number of items in the search list is probably limited by the workspace available, and the amount of data carried in the macro variables. I have successfully run lists with over 75,000 search keys. Table methods using PROC FORMAT will probably run a little faster than the 'ON THE FLY' method, but will not work in the second example above. The nice thing about the 'ON THE FLY' method is it's simplicity.  Only one line of simple to understand code is required and can be placed just about anywhere and whenever you want to use it.