Paper 065-2010

Space Management for Text Variables

William C. Murphy
Howard M. Proskin & Associates, Inc.

ABSTRACT

Spaces in text strings are vitally important. Without spaces between words, text would be completely unreadable and impossible to align. However, too many spaces between the words would leave the text just as difficult to read. The statements and functions of the SAS® system sometimes insert or remove spaces in a surprising way. Fortunately, SAS also provides us with functions such as TRIM and STRIP to remove unwanted spaces. Furthermore with the proper choice of functions and options, excess spaces need not be generated. On the other hand, retaining spaces in variables can be just as much of a problem. For instance, in creating macro variables with the %LET statement, careful consideration must be given to retain spaces before or after the character string. Careful attention must also be paid to text variable created with the concatenation functions. Various functions and statements such as those already mentioned and others, such as PUT and SUBSTR, will be examined to insure the proper maintenance and control of spaces in text strings.

INTRODUCTION

Spaces are a necessary tool for the creation of textual material. Without spaces defining the limits of words and sentences, there are no words or sentences. Nevertheless, too many spaces are not only wasteful in printed material, but make reading words and sentences extremely difficult. As a programmer, we find spaces constantly occurring in variables containing text strings. When we use SAS to create or manipulate these text variables, we sometimes inadvertently insert new spaces into the strings. Other times we suppress spaces that should be there. We will review the various SAS functions and statements for their handling of spaces, and make suggestion to improve space management in text strings.

REMOVING THE UNWANTED

The SAS system has several great ways of eliminating unwanted space. Murphy (2006) showed that the ultimate way to remove all spaces (and anything else) from a string is the COMPRESS function. However, sometimes it is just the leading or trailing spaces on a string that are a problem. For old timers, the spaces in front of a character string can be removed with the LEFT function. The TRIM function can also be applied to remove any excess spaces to the right of the character string. Together, these functions will remove all unwanted spaces before and after character strings. If you are using SAS version 9, this operation can me more easily accomplished with the STRIP function. However, creating unneeded spaces before or after text strings should not be done in the first place. The LEFT and TRIM functions or the STRIP functions should only be used when these spaces are unavoidable. The PUT statement is a prime example where spaces can, but need not be generated.

THE PUT STATEMENT

For those using a DATA step, the put statement is an extremely useful tool. You can use it not only for debugging but for writing custom output. However, if you output several text variables in a row, the PUT has the annoying habit of inserting spaces between the strings whether you want them there or not. For example, if you want to create words from individual letters in the PUT output, you might write

```
LetterS = 'S';
LetterA = 'A';
put  "I am a " LetterS LetterA LetterS " programmer";
```

I would get

I am a S A S programmer.

Because whenever you write variables with the PUT, the SAS system inserts a space before each string, except for the first variable string. If we don't want these spaces we have to insert code to remove them:

```
put  "I am a " LetterS +(-1) LetterA +(-1) LetterS +(-1) " programmer";
```

which produces

I am a SAS programmer.

Although the code may look odd, the results are exactly what we want. The '+' sign here is not an indication of a positive value or addition, but rather an invocation of the put space control. The '+' tell PUT to move the string the indicated number of spaces. In this case, the '-1' moves the string one space to the left, effectively removing the automatic space that SAS inserts.

THE PUT FUNCTION

Just as the PUT statement inserts spaces that are sometimes unwanted, so does the PUT function. If you ever had to convert some data from numeric to character, you might have written

```
Score = 123;
AlphaScore = put(Score,5.);
```

But every time you do some type of test for the string '123' in AlphaScore it fails. Similar to the statement, the PUT function is inserting a leading space. Now we could use LEFT and TRIM or the STRIP function to get ride of this, but wouldn't it be better not to create the problem in the first place. We can no longer use the '+' option, but the function does allow the insertion of justification;

```
AlphaScore = put(Score,5. -L);
```

where the –L after the format shifts the string to the left and eliminates the leading space.

ANOTHER 'PUT' PROBLEM

Most of us at one time or another have generated macro variable in a DATA step:

```
data example;
       set sashelp.class;
       call symput(cats('Name',_n_),name);
       run;
```

In this example, we have created macro variables &Name1 through &Name19 that hold the names of the subjects in the sample data set class. If we wish to display the value contained in the first macro variable, we might write

```
%put &Name1 is the first person;
```

which will produce

Alfred is the first person

Why there are extraneous spaces after 'Alfred'? CALL SYMPUT pads the character string with spaces up to the allowed length of the variable (in this case 8). What is the solution? Simple, don't use CALL SYMPUT. The SAS system now has a new function, CALL SYMPUTX which acts nearly identically to CALL SYMPUT but does not pad any spaces on the left of the variable.

SPACE CONFUSION

As Csont (2008) pointed out, sometimes you get spaces where you think data should be. For example, we received some information for a group of patients on the medications they were taking. If the patient wasn't asked or missed the question the answer was left blank. However, if the interviewer knew the patient was on a medication but didn't know what type, a '.' would be inserted into the response. So the raw data was read into SAS:

```
data Meds;
       infile datalines missover;
       informat med $10.;
       input PatientNo Med;
       datalines;
       1  Tums
```

```
2  AklaSeltzer
3  .
4
5 Rollaids
;
```

If we examine the newly created SAS data set Meds, we find two missing values! SAS replaced the '.' with a space. If you don't want this space and want the system to read the characters that are actually there, you cannot use the $10. informat. Instead write

```
informat meds $char10.;
```

where the informat Char treat '.' as '.' and not a space.

SPACE REORDERING

If you are given 100 lines of data consisting of names (first, middle, last) in a Microsoft Excel spreadsheet, you could easily read this data into the SAS system by writing

```
filename spread dde 'Excel|Sheet1!r1c1:r100C3';

data SheetData;
       infile spread missover;
       length FName MName LName $15;
       input FName MName LName;
       run;
```

This code would work perfectly as long as you did not have a blank in Column A or Column B in your spreadsheet. If you had such a space, the SAS system would read the line as if the spaces where at the end of the row! So if you read in a spreadsheet row with the values of column A = Bilbo, column B = *{blank}*, and column C = Baggins, the SAS system would read FName = 'Bilbo', MName = 'Baggins', and LNmane=' '. This occurs because of the way SAS treats the default tab delimiter in the spreadsheet, as was explained by Vyerman(2002). To overcome this problem we rewrite the code as

```
filename spread dde 'Excel|Sheet1!r1c1:r100C3' notab;

data SheetData;
       infile spread delimiter='09'x dsd missover;
       length FName MName LName $15;
       input FName MName LName;
       run;
```

where we initially strip away the delimiters (NOTAB) and then reinsert them (DELIMITER=), and make sure that spaces are treated as missing (DSD). This will work perfectly on reading the data. Since you never know when a space will appear in your data, these options should always be used to prevent the spaces from being repositioned.

PUTTING WORDS TOGETHER

When we use the old concatenation operator, ||, to combine words into larger string, we invariably end up with too few or too many spaces. We manipulated the individual strings with LEFT and/or TRIM and maybe even a space string, ' ', to obtain the desired results. Although the || operator is still in the SAS system, it has functionally been replaced by the CAT, CATT, CATS, and CATX functions. Now CAT works nearly identically to a series of || operators and CATT is not much better. Almost all concatenation can be done using the CATS and CATX. To illustrate we write in a DATA step the following code

```
Alpha = 'Code              ';
Beta = '         Not';
Gamma = 'Spaces';

Delta = cats(Alpha, Beta,Gamma);
Epsilon = catx(' ',Alpha,Beta,Gamma);

put Delta = /Epsilon = ;
```

which will produce in the log

```
Delta =CodeNotSpaces
Epsilon =Code Not Spaces
```

So both the CATS and CATX functions get rid of all preceding and trailing spaces. But the CATX function inserts the first string (in this case a space) between each of its arguments. Judicious use of the CATS and CATX solves most concatenation induced space problems.

STREAMING BLANKS

For unformatted data, reading one byte at a time is the way that I get my input:

```
data one;
       length value $200;
       retain value ' ' LineCnt 1 PreviousEndCol 0 BlankCnt 0 ColCnt 1;
       infile raw recfm=n ;
       input InChar $char1.;
```

I get the information that I need from this data by concatenation of several of the InChar variables in a certain sequence, defined by the source of the unformatted data. If the InChar contains a space the CATS and CAT function will remove them. Instead I use the SUBSTR function on the left-hand of the equal to build my character strings:

```
       length Value $1000;

       LenValue = length(Value);
       Position = LenValue+1+BlankCnt;
       substr(Value,Position,1)=InChar;
       if InChar = ' ' then BlankCnt+1;
       else BlankCnt=0;
```

where each time through the DATA step, the InChar is added to the Value by the SUBSTR function at the calculated Position. Position is determined by the previous size of the Value string along with a BlankCnt which keeps track of the spaces. This method is quite effective for managing the spaces in the input data stream.

NOT ENOUGH SPACE

We have already discussed the space problem with macro variables created in a DATA step, but problems can also arise with the %LET statement. If we write

```
%let Alpha =   Code   ;
%let Beta  =   Not    ;
%let Gamma =   Spaces   ;

%put &Alpha&Beta&Gamma;
```

where the LOG would display

CodeNotSpaces

The SAS system ignores all spaces, preceding and trailing, in the %LET statement. If we really want the spaces there, we must use a macro quoting function:

```
%let Alpha = %str(   )Code%str(   );
%let Beta  = %str(   )Not%str(   );
%let Gamma = %str(   )Spaces%str(   );
```

The %PUT statemement will now produce

Code Not Spaces

where all the spaces contained in the %STR are retained.

CONCLUSIONS

Spaces can be an annoyance to the SAS programmer, but keeping the spaces down to a desirable number is an absolute necessity. By careful use of appropriate functions, such as CATX and CATS, over other concatenation operators, along with the certain options on functions, such as PUT, spaces can be controlled in many cases. You can even control the placing of spaces in text strings by using the SUBSTR function. Furthermore, using the simple functions of TRIM and STRIP and the macro function %STR allows the programmer additional tools for space management in text variables.

REFERENCES

Csont, W.C. (2008), "When a Period Really is Not a Period and How to Fix It", *Proceedings of the 2008 Annual Conference of the Pharmaceutical Industry SAS Users Group,* paper CC19 (url: http://www.lexjansen.com/pharmasug/2008/cc/cc19.pdf).

Murphy, W.C. (2006), "Squeezing Information out of Data", *Proceedings of the Thirty-First Annual SAS® Users Group International Conference*, SAS Institute Inc., Cary, NC, paper 028-31 (url: http://www2.sas.com/proceedings/sugi31/028-31.pdf).

Vyverman, K. (2001), "Using Dynamic Data Exchange to Export Your SAS Data to MS Excel", Proceedings of the Twenty-Sixth *Annual SAS® Users Group International Conference*, SAS Institute Inc., Cary, NC, paper 11 (url: http://www2.sas.com/proceedings/sugi26/p011-26.pdf).

CONTACT INFORMATION

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

William C. Murphy

Howard M. Proskin & Associates, Inc.

300 Red Creek Dr., Suite 220

Rochester, NY 14623

Phone 585-359-2420

FAX 585-359-0465

Email wmurphy@hmproskin.com or wcmurphy@usa.net

Web www.hmproskin.com