**PharmaSUG 2014 - Paper TT12**

# Let the CAT Out of the Bag: String Concatenation in SAS 9

Joshua Horstman, Nested Loop Consulting, Indianapolis, IN

## ABSTRACT

Are you still using TRIM, LEFT, and vertical bar operators to concatenate strings? It's time to modernize and streamline that clumsy code by using the string concatenation functions introduced in SAS 9. This paper is an overview of the CAT, CATS, CATT, and CATX functions introduced in SAS 9.0, and the new CATQ function added in version 9.2. In addition to making your code more compact and readable, this family of functions also offers some new tricks for accomplishing previously cumbersome tasks.

## INTRODUCTION: STRING CONCATENATION BEFORE SAS 9

String concatenation refers to the joining of two or more character strings into a single, longer string. This is a common task performed frequently by most programmers. As such, it's important for the programmer to have the best tools at his or her disposal for accomplishing this task.

Prior to the release of version 9 of the SAS software in 2002, string concatenation was typically performed using the double vertical bar operator (||). This operator is frequently used in conjunction with the TRIM and LEFT functions in order to remove any leading or trailing blank characters. The LEFT function left-aligns a string by moving leading blanks to the end. The TRIM function then removes all trailing blanks. Thus, it is common to see SAS code of the following form:

```
newvar = trim(left(var1))||trim(left(var2))||trim(left(var3));
```

While there is nothing wrong with this code, it can become unwieldy to read and edit. Fortunately, SAS 9 provides new options for accomplishing this rather mundane task. In this paper, we will review some of the new functions available in SAS 9 that facilitate string concatenation. We will see that these new methods are often more convenient and produce more compact code that is easier to understand and maintain. We will also discuss examples where the clever use of these techniques can simplify code that was previously cumbersome.

## NEW FUNCTIONS IN SAS 9

### THE STRIP FUNCTION

One new function introduced in SAS 9 is the STRIP function. The STRIP function removes both leading and trailing blanks from a string, thus combining the functionality of TRIM and LEFT. This allows us to simplify the code above to the following:

```
newvar = strip(var1)||strip(var2)||strip(var3);
```

The STRIP function gives us an incremental improvement over using TRIM and LEFT. Technically, `STRIP(var)` returns the same result as `TRIMN(LEFT(var))`. This is slightly different from `TRIM(LEFT(var))` if `var` is a blank string because the TRIM function will return a single blank. In contrast, the STRIP function, like the TRIMN function, will return a zero-length string.

### THE CAT FAMILY OF FUNCTIONS

A more significant addition to SAS 9 is the CAT family of functions. These functions include CAT, CATS, CATX, and CATT, all available in version 9.0, along with CATQ, which came along in SAS 9.2. We'll look briefly at each one first and then discuss the advantages and limitations of using these functions.

**CAT:** The CAT function is the most basic string concatenation function. It takes any number of character strings as arguments and simply joins them all end-to-end. The result is the same as that obtained using the double vertical bar operator: `CAT(var1, var2, var3)` is equivalent to `var1||var2||var3`.

**CATT:** The CATT function is similar to the CAT function, but it removes any trailing blanks from each string before joining them together. This is equivalent to calling the TRIM function on each argument prior to concatenation, so it may be helpful to think of the final T in CATT as an abbreviation for TRIM. Thus, `CATT(var1, var2, var3)` is equivalent to `TRIM(var1)||TRIM(var2)||TRIM(var3)`.

**CATS:** The CATS function is another variation on CAT in which both leading and trailing blanks are removed before the arguments are concatenated. The result is the same that would be obtained by invoking the STRIP function on each argument before joining them, so one can think of the final S in CATS as standing for STRIP. As discussed above, the STRIP function is equivalent to calling TRIMN and LEFT. The corresponding syntax for `CATS(var1, var2, var3)` prior to SAS 9 would be `TRIMN(LEFT(var1))||TRIMN(LEFT(var2))||TRIMN(LEFT(var3))`.

**CATX:** The CATX function goes one step further than CATS. In addition to removing leading and trailing blanks, CATX will also insert a specified delimiter before concatenating the other arguments. The delimiter is the first argument to CATX. One might consider the X in CATX as indicating that the function inserts something "eXtra".

In general, one can consider `CATX(' ',var1, var2, var3)` as being equivalent to `TRIMN(LEFT(var1))||' ' ||TRIMN(LEFT(var2))||' '||TRIMN(LEFT(var3))`. However, this is not completely true because CATX is a bit more sophisticated. While the traditional syntax will produce consecutive duplicate delimiters when dealing with blank strings, CATX will automatically eliminate the redundant delimiters.

**CATQ:** The CATQ function is a veritable Swiss army knife of string concatenation functions. Its default functionality is similar to that of CATX, but the CATQ function adds quotation marks to any string that contains the specified delimiter. The Q in CATQ might be thought of as referring to quotation marks.

However, the CATQ offers numerous options to modify its behavior, including the ability to remove leading and/or trailing blanks before concatenation. A full discussion of the various features of CATQ is beyond the scope of this paper. Refer to the SAS documentation for more details.

## ADVANTAGES OF USING CAT FUNCTIONS

The CAT functions offer several advantages over the traditional syntax commonly used prior to version 9. As can be seen above in the descriptions of each function, the syntax is typically more compact and easier to read. This can be a great help to someone unfamiliar with the code who needs to understand what it does and possibly make changes.

In addition, the CAT functions offer several additional conveniences to the programmer. First, all of the CAT functions support the OF syntax. This shorthand notation allows the programmer to refer to an entire group of sequentially numbered variables without listing each one. For example, instead of writing `X1||X2||X3||X4||X5` or even `CAT(X1,X2,X3,X4,X5)`, the programmer can simply write `CAT(OF X1-X5)`.

Secondly, the CAT functions eliminate the need to write complex logic to prevent duplicate delimiters that come about because of blank strings. As mentioned above, the CATX and CATQ functions handle this automatically.

Finally, numeric values which are passed as arguments to any of the CAT functions are automatically converted to strings without any additional notes to the log. This conversion is equivalent to using the PUT function with a BEST12 format. While the traditional concatenation operators will also perform this automatic conversion, they will generate a "NOTE: Numeric values have been converted to character values …" in the log unless explicitly converted (e.g. using the PUT function). This can be important when a clean SAS log is necessary.

## NOTE ABOUT VARIABLE LENGTHS

Another important way in which the CAT functions differ from the traditional syntax has to do with the handling of variable lengths. When a concatenation operator is used, the variable returned has a length equal to the sum of the

lengths of the values being concatenated, unless otherwise specified. In contrast, the CAT functions always return a variable with a length of 200 unless the variable was previously assigned another length. In certain circumstances, this can cause the results to differ between the two methods. In either case, it's important to check the attributes of variables being created to ensure that data is not being truncated due to insufficient length.

## USING THE NEW FUNCTIONS TO SIMPLIFY YOUR CODE

We will now discuss two examples in which clever use is made of the CAT family of functions to simplify the code. The first example is adapted from one given in a 2012 paper by Marje Fecht, and the second is based on ideas from a 2009 paper by Mike Zdeb (see references below).

### EXAMPLE #1: THE SETUP

Consider the clinical laboratory results data below. For brevity, only one subject and three lab tests are included.

LAB Dataset

| SUBJECT | LABTEST | HIGHFLAG | LOWFLAG | BLFLAG | CSFLAG | WPBFLAG |
|---|---|---|---|---|---|---|
| 1 | Bilirubin | 1 | 0 | 1 | 0 | 0 |
| 1 | Calcium | 0 | 0 | 0 | 0 | 0 |
| 1 | Potassium | 0 | 1 | 0 | 1 | 1 |

For this example, we aren't interested in the actual lab result, but only in a series of flag variables, each of which holds a 0 or 1 value:

HIGHFLAG – Value of 1 indicates the lab result is high (i.e. above the normal range)
LOWFLAG – Value of 1 indicates the lab result is low (i.e. below the normal range)
BLFLAG – Value of 1 indicates the lab result represents a baseline value (i.e. prior to treatment)
CSFLAG – Value of 1 indicates the lab result is clinically significant
WPBFLAG – Value of 1 indicates the lab result is the worst post-baseline value.

The goal of this exercise is to create a new variable, FLAGLIST, which lists the applicable flags for each record, separated by commas, to be used in a report. In this list, the flags are to be coded as H, L, B, CS, and WPB, respectively. That is, FLAGLIST should have a value of "H,B" for the first record, should be blank for the second record, and should contain "L,CS,WPB" for the third record.

### EXAMPLE #1: OLD SCHOOL SOLUTION

There are many ways to solve this problem. Below is one way a programmer might have commonly handled this task prior to SAS 9. Notice the extensive logic pertaining to the placement of commas. This is necessary to ensure that commas do not appear at the start or end of the list and that multiple commas do not appear between flag codes.

```
data lab2a;
   set lab;
   need_comma = 0;
   length flaglist $12;
   if highflag then do;
      flaglist = 'H';
      need_comma = 1;
   end;
   if lowflag then do;
      if need_comma then flaglist = trim(left(flaglist))||',';
      flaglist = trim(left(flaglist))||'L';
      need_comma = 1;
   end;
   if blflag then do;
      if need_comma then flaglist = trim(left(flaglist))||',';
      flaglist = trim(left(flaglist))||'B';
      need_comma = 1;
   end;
   if csflag then do;
      if need_comma then flaglist = trim(left(flaglist))||',';
      flaglist = trim(left(flaglist))||'CS';
      need_comma = 1;
   end;
```

```
   if wpbflag then do;
      if need_comma then flaglist = trim(left(flaglist))||', ';
      flaglist = trim(left(flaglist))||'WPB';
   end;
run;
```

Below is a brief report generated using the dataset created above. The FLAGLIST variable was populated correctly.

```
proc print data=lab2a noobs;
       var subject labtest flaglist;
run;
```

| subject | labtest | flaglist |
|---|---|---|
| 1 | Bilirubin | H,B |
| 1 | Calcium | |
| 1 | Potassium | L,CS,WPB |

## EXAMPLE #1: MODERNIZED SOLUTION

Take a look at the alternative version below. This code produces the exact same results, but is dramatically shorter. Not only have we reduced 28 statements to 4, but we've vastly simplified the logic and eliminated the need for an extra variable.

This is accomplished through the use of the CATX and IFC functions. The IFC function, also new in SAS 9, is not a string concatenation function, but it is often a handy method for conditionally determining the value of a character variable. Note that the IFC function passes a blank string into CATX whenever one of the flag values is 0 and CATX conveniently only adds commas between the non-blank strings corresponding to the flag values of 1.

```
data lab2b;
   set lab;
   flaglist = catx(',',
      ifc(highflag,'H'  ,''),
      ifc(lowflag ,'L'  ,''),
      ifc(blflag  ,'B'  ,''),
      ifc(csflag  ,'CS' ,''),
      ifc(wpbflag ,'WPB','')
      );
run;
```

The output below confirms that this method produces the same results as the traditional code above.

```
proc print data=lab2b noobs;
       var subject labtest flaglist;
run;
```

| subject | labtest | flaglist |
|---|---|---|
| 1 | Bilirubin | H,B |
| 1 | Calcium | |
| 1 | Potassium | L,CS,WPB |

## EXAMPLE #2: THE SETUP

Our second example deals with the responses to a series of a yes/no questions posed to potential subjects in a clinical trial. Any affirmative response is considered a criterion to exclude the subject from participation in the trial. We wish to produce a list of subjects who have responded positively to one or more questions. Based on the data below, our list should include subjects 1 and 3.

EXCLUSION_DATA Dataset

| SUBJECT | E1 | E2 | E3 | E4 | E5 |
|---|---|---|---|---|---|
| 1 | N | N | N | N | Y |
| 2 | N | N | N | N | N |
| 3 | Y | Y | N | Y | Y |

## EXAMPLE #2: OLD SCHOOL SOLUTION

Again, we'll start with a traditional approach to this task. The code below creates an array of the five response variables and then loops through the array for each record until a "Y" is found. The record is written to the output dataset only if the loop terminates early, which means a "Y" response was encountered

```
data excluded_subjects;
        set exclusion_data;
        array e(5);
        do i=1 to 5 until (e(i) = 'Y');
        end;
        if i < 6;
        drop i;
run;
```

The output below includes only subjects 1 and 3 as expected.

```
proc print data=excluded_subjects noobs;
        var subject;
run;
```

| subject |
|---|
| 1 |
| 3 |

## EXAMPLE #2: MODERNIZED SOLUTION

The modernized code below uses the CAT function and the OF syntax in conjunction with the FIND function (also new in SAS 9). In a single statement, all the responses for a given record are concatenated and the row is written to the output dataset only if the combined string contains a "Y".

```
data excluded_subjects2;
        set exclusion_data;
        if find(cat(of e:),'Y');
run;
```

Notice that this code would require no changes if additional responses were added later, so long as the existing naming convention were followed. The traditional code above would have to be modified based on the number of responses. (It would be possible to alter the traditional code to automatically detect the number of responses, but doing so would make the code even more complex.)

Once again, we generate a simple report to confirm that the code has produced the desired result.

```
proc print data=excluded_subjects2 noobs;
        var subject;
run;
```

| subject |
|---|
| 1 |
| 3 |

## CONCLUSIONS

The CAT family of functions offers tremendous convenience to the SAS programmer. Used properly, these functions can simplify and shorten code while making it easier to understand and modify. The CAT functions also offer some new ways to solve old problems.

This is not to say that these functions are the right choice in all situations. The traditional double vertical bar operator and legacy functions such as TRIM and LEFT continue to be a useful part of the SAS programmer's collection of tools. The savvy SAS programmer will be well-versed in the use of all of the above and will select the most appropriate tool based on the task at hand.

## REFERENCES


Fecht, Marje. Quick Hits - My Favorite SAS® Tricks. Proceedings of the SouthEast SAS Users Group, Durham, NC, 2012.

Hadden, Louise S. Purrfectly Fabulous Feline Functions. Proceedings of the SAS® Global Forum 2010 Conference. Cary, NC: SAS Institute Inc., 2010.

SAS Institute Inc. SAS® 9.4 Functions and CALL Routines: Reference. Cary, NC: SAS Institute Inc., 2013.

Zdeb, Mike. Searching for Variable Values with CAT Functions: An Alternative to Arrays and Loops. Proceedings of the NorthEast SAS Users Group, Burlington, VT, 2009.


## CONTACT INFORMATION

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


Name: Joshua Horstman
Enterprise: Nested Loop Consulting
Phone: 317-815-5899
E-mail: jmhorstman@gmail.com