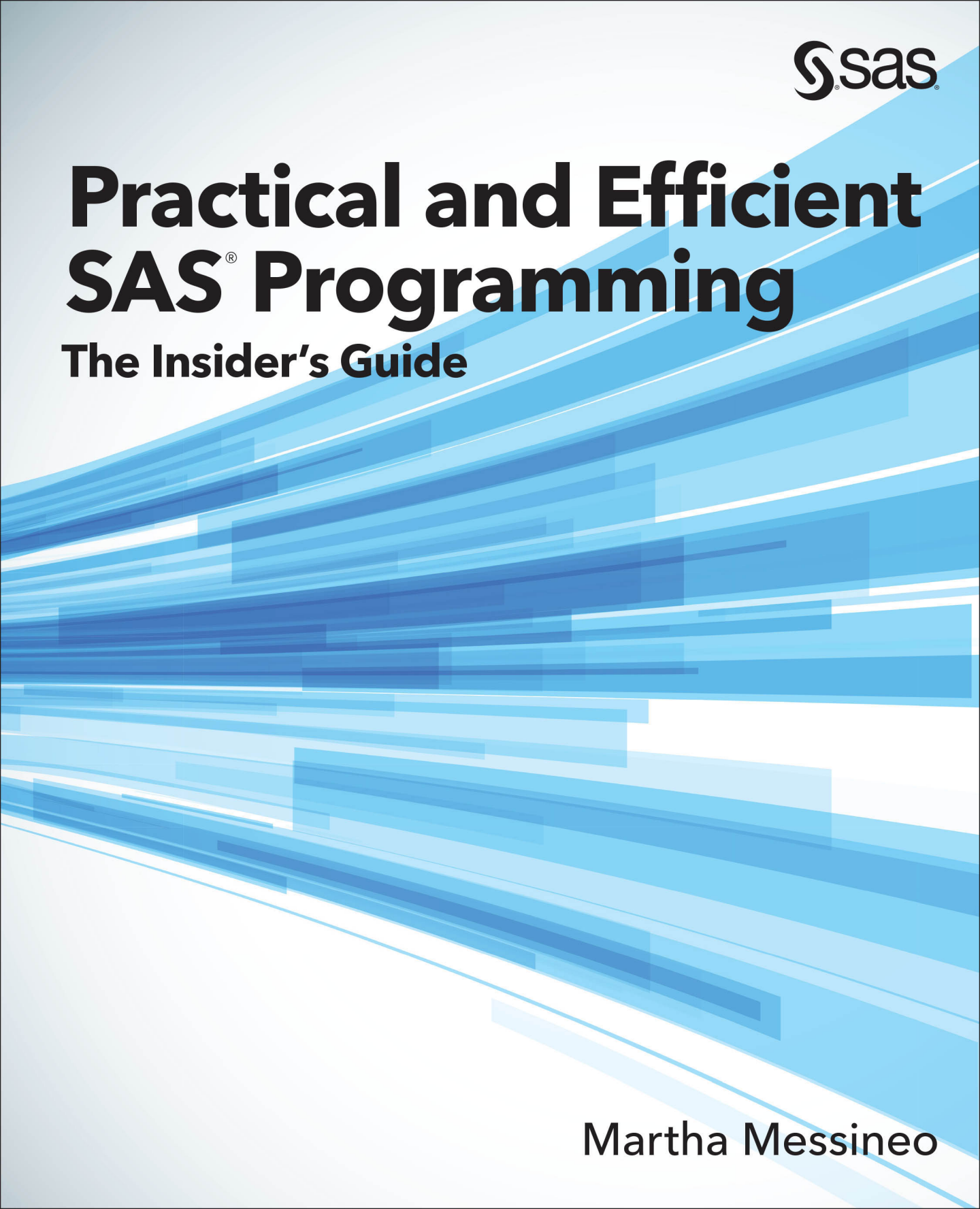

sas
Practical and Efficient SAS® Programming
The Insider's Guide
Martha Messineo

The correct bibliographic citation for this manual is as follows: Messineo, Martha. 2017. *Practical and Efficient SAS® Programming: The Insider's Guide*. Cary, NC: SAS Institute Inc.

Practical and Efficient SAS® Programming: The Insider's Guide

Copyright © 2017, SAS Institute Inc., Cary, NC, USA

ISBN 978-1-63526-023-6 (Hard copy)
ISBN 978-1-63526-222-3 (EPUB)
ISBN 978-1-63526-223-0 (MOBI)
ISBN 978-1-63526-224-7 (PDF)

All Rights Reserved. Produced in the United States of America.

For a hard copy book: No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

For a web download or e-book: Your use of this publication shall be governed by the terms established by the vendor at the time you acquire this publication.

The scanning, uploading, and distribution of this book via the Internet or any other means without the permission of the publisher is illegal and punishable by law. Please purchase only authorized electronic editions and do not participate in or encourage electronic piracy of copyrighted materials. Your support of others' rights is appreciated.

U.S. Government License Rights; Restricted Rights: The Software and its documentation is commercial computer software developed at private expense and is provided with RESTRICTED RIGHTS to the United States Government. Use, duplication, or disclosure of the Software by the United States Government is subject to the license terms of this Agreement pursuant to, as applicable, FAR 12.212, DFAR 227.7202-1(a), DFAR 227.7202-3(a), and DFAR 227.7202-4, and, to the extent required under U.S. federal law, the minimum restricted rights as set out in FAR 52.227-19 (DEC 2007). If FAR 52.227-19 is applicable, this provision serves as notice under clause (c) thereof and no other notice is required to be affixed to the Software or documentation. The Government's rights in Software and documentation shall be only those set forth in this Agreement.

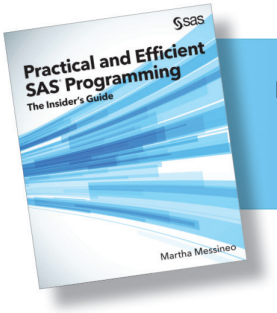
SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

September 2017

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.

SAS software may be provided with certain third-party software, including but not limited to open-source software, which is licensed under its applicable third-party software license agreement. For license information about third-party software distributed with SAS software, refer to **http://support.sas.com/thirdpartylicenses**.



Np_arga_j _lb Cddgagclr Q?Q$ Npmep_k kglel Rfc Glqgbcp¦q
Esgbc. Full book available for purchase here.

Contents

From Practical and Efficient SAS® Programming: The Insider's Guide, by Martha Messineo. Copyright © 2017, SAS Institute Inc., Cary, North Carolina, USA. ALL RIGHTS RESERVED.

Chapter 1: My Favorite Functions

Introduction

There are literally hundreds of DATA step functions. It is so easy to get overwhelmed by the sheer volume that you miss can some real gems. Here are some functions that I can't live without and some different ways to use them.

Concatenating Strings

If you need to combine two or more strings into a single string, you can use the concatenation operator: ||. For example, you can take two strings such as "Mary" and "Smith" and create a new string with a value of "Mary Smith":

```
newString = "Mary" || " " || "Smith";
```

Tip: You might see SAS code that uses !! instead of || as the concatenation operator. Both are valid operators. The reason that some people use the !! is because several years ago (okay … many years ago), mainframe keyboards didn't have the | character. So, if you see !! instead of ||, it probably means that the coder originally learned SAS on a mainframe.

The CAT() functions—CAT(), CATS(), CATX(), CATQ(), and CATT()—can also be used to concatenate strings together. So why not just use the standard || operator to do concatenation? The CAT() functions do additional work including stripping leading and trailing spaces and converting numbers to characters. I still use the || syntax at times, but I use the CAT() functions more.

So if you are doing basic concatenation, you can simplify the following syntax:

```
newString = trim(left(string1)) || left(string2);
```

Instead, you can use the following syntax, which is much easier to type and read:

```
newString = cats(string1, string2);
```

CATX() adds delimiters between your strings. You can simplify the following syntax:

```
newString = trim(left(string1)) || " " || left(string2);
```

Instead, you can use this:

```
newString = catx(" ", string1, string2);
```

Converting Numbers to Characters

The ability of these functions to do basic concatenation makes them great tools, but you can do more.

The CAT() functions also convert numeric values into character strings, so you don't have to use a PUT() function. So to concatenate a number to a string, you could do this:

```
newString = trim(left(string1)) || left(put(number, 10.));
```

But this is much simpler:

```
newString = cats(string1, number);
```

When converting a number to a string without using the CAT() function, you need to use the PUT() function and specify the format that you want. So you might do this:

```
newString = left(put(number, 3.));
```

But what happens if you miscalculate, and the value for number is bigger than 999? SAS software fits the value into 3 characters, so the result is "1E3" instead of "1000". Using the CATS() function, you don't have to worry about the length of the actual number, and you can just do the following:

```
newString = cats(num);
```

The example in Program 1.1 and SAS Log 1.1 shows how to convert a number using the traditional style, and then using the CATS() function.

Program 1.1: Converting a Number to a String

```
data _null_;
   length newString_put newString_cat $10;
   number = 1000;

   newString_put = left(put(number, 3.));
   putlog newString_put=;

   newString_cat = cats(number);
   putlog newString_cat=;
run;
```

SAS Log 1.1: Converting a Number to a String

```
newString_put=1E3
newString_cat=1000
```

Tip: If you are converting a number to a character and you need the new string to look a certain way—maybe with a $ or commas—you need to use a PUT() function with the appropriate format. Use the CAT() functions only to convert simple numbers into simple strings.

Adding Delimiters

At some point, you might have had to create a string that consists of a series of other strings separated by a delimiter (such as a blank or a comma). When you do this without CATX(), you need to use syntax similar to this:

```
data _null_;
   set sashelp.class end = eof;
   where sex eq "F";

   length girlList $500;
   retain girlList "";

   if (_n_ eq 1) then
      girlList = left(name);
   else
      girlList = trim(left(girlList)) || ", " || left(name);

   if (eof) then
      putlog girlList=;
run;
```

This syntax produces the following output:

```
girlList=Alice, Barbara, Carol, Jane, Janet, Joyce, Judy, Louise,
Mary
```

This code does what it should: It puts all the girls' names in a comma delimited list. But it needs additional code to avoid putting an extra comma at the beginning (as in this example) or end of the list.

An alternative is to use CATX(). Then you don't have to worry about extra commas, because CATX() only puts the delimiter between non-blank values. Program 1.2 and SAS Log 1.2 show how to use the CATX() function to add a comma between the girls' names.

Program 1.2: Concatenating Strings with a Delimiter

```
data _null_;
   set sashelp.class end = eof;
   where sex eq "F";

   length girlList $500;
   retain girlList "";

   girlList = catx(", ", girlList, name);

   if (eof) then
      putlog girlList=;
run;
```

SAS Log 1.2: Concatenating Strings with a Delimiter

```
girlList=Alice, Barbara, Carol, Jane, Janet, Joyce, Judy, Louise,
Mary
```

Another way that CATX() is useful is when you want to concatenate several strings with a delimiter, but you don't want extra delimiters if some of the strings are blank. For example, suppose you have a first, middle, and last name, and you want to create a full name out of the parts. If the middle name is blank, you don't want to add extra blanks. If you use ||, then you must check to see whether the middle name is blank, or you will end up with too many blanks in the name.

The code in Program 1.3 creates a data table to use with the example in Program 1.4, and Figure 1.1 shows a listing of the table.

Program 1.3: Creating the Names Data Table

```
data names;
   input @1 first $5. @7 middle $5. @15 last $10.;
   datalines;
Sue            Jones
Ann   K.       Smith
Joe            Thomas
Sally Jo       Anderson
;
```

Figure 1.1: Creating the Names Data Table

Obs	first	middle	last
1	Sue		Jones
2	Ann	K.	Smith
3	Joe		Thomas
4	Sally	Jo	Anderson

You could just concatenate the 3 names like this:

```
name = strip(first) || " " || strip(middle) || " " || strip(last);
```

You would get these names as a result:

```
Sue Jones
Ann K. Smith
Joe Thomas
Sally Jo Anderson
```

You can see that both Sue and Joe have an extra space because they don't have middle names.

To deal with this issue, you can test the middle name to see whether it is blank before adding it to the name:

```
if (middle eq "") then
   name = strip(first) || " " || strip(last);
else
   name = strip(first) || " " || strip(middle) || " " ||
          strip(last);
```

The names then have the correct spacing:

```
Sue Jones
Ann K. Smith
Joe Thomas
Sally Jo Anderson
```

Tip: You could use the COMPBL() function to take care of the extra spaces. It replaces multiple spaces with a single space.

Using CATX(), you don't have to worry about extra blanks. If a value is missing, CATX() doesn't add the extra delimiter, as you can see in the example in Program 1.4 and SAS Log 1.3.

Program 1.4: Using CATX() to Avoid Checking for a Blank Value

```
data _null_;
   set names;
   length name $50;
   name = catx(" ",
               first, middle, last);
   putlog name=;
run;
```

SAS Log 1.3: Using CATX() to Avoid Checking for a Blank Value

```
name=Sue Jones
name=Ann K. Smith
name=Joe Thomas
name=Sally Jo Anderson
```

Using the OF Shortcut

In addition to all the benefits of the CAT() functions, you can also use the OF shortcut to concatenate a series of strings.

If you want to create a unique key variable on each record of the table, then you can concatenate all the variables with a delimiter between the values. Or maybe you want to create a comma-delimited file out of all the variables in the table. If you don't use CATX(), then it is a complicated task, as the following program illustrates:

```
proc contents data = sashelp.class
              out = contents (keep = name varnum)
              noprint nodetails;
run;

data _null_;
   set sashelp.class;
   length newString $1000;
   newString = "";
   do p = 1 to nobs;
      set contents point = p nobs = nobs;
      newString = strip(newString) || "," ||
                  strip(vvaluex(c_name));
   end; /* do p - loop through contents */
   /* get rid of beginning comma */
   newString = substr(newString, 2);
run;
```

Tip: If you have the name of a variable in another variable, use the VVALUEX() function to get the first variable's value:

```
x = "y";
y = "abc";
z = vvaluex(x);
putlog x= y= z=;
```

Output: x=y y=abc z=abc

Program 1.5 and SAS Log 1.4 show how to use the CATX() function with the OF _ALL_ shortcut to simplify adding all the variable values together.

Program 1.5: Using CATX() to Concatenate All Strings

```
data _null_;
   set sashelp.class;
   length newString $1000;
   newString = catx(",", of _ALL_);
   putlog newString=;
run;
```

SAS Log 1.4: Using CATX() to Concatenate All Strings

```
newString=Alfred,M,14,69,112.5
newString=Alice,F,13,56.5,84
newString=Barbara,F,13,65.3,98
newString=Carol,F,14,62.8,102.5
newString=Henry,M,14,63.5,102.5
newString=James,M,12,57.3,83
newString=Jane,F,12,59.8,84.5
newString=Janet,F,15,62.5,112.5
newString=Jeffrey,M,13,62.5,84
newString=John,M,12,59,99.5
newString=Joyce,F,11,51.3,50.5
newString=Judy,F,14,64.3,90
newString=Louise,F,12,56.3,77
newString=Mary,F,15,66.5,112
newString=Philip,M,16,72,150
newString=Robert,M,12,64.8,128
newString=Ronald,M,15,67,133
newString=Thomas,M,11,57.5,85
newString=William,M,15,66.5,112
```

Removing Leading and Trailing Spaces

When you need to remove spaces from the beginning and end of a string, the STRIP() function can take care of it for you. It is a great function because it replaces two functions: LEFT() and TRIM(). How could you not love that?

The STRIP(*string*) function is almost the same as TRIM(LEFT(*string*)). The only difference is that if you use STRIP() on a blank string, it returns a value of zero characters, while TRIM() returns a string of one blank character. Using TRIMN(LEFT(*string*)) returns a zero-length string just as the STRIP() function does.

Program 1.6 and SAS Log 1.5 compare the results from using the LEFT() and TRIM() functions to using the STRIP() function.

Program 1.6: Removing Spaces

```
data _null_;
   x1 = " abc  ";
   x2 = "    ";
   array x {2};
   do i = 1 to dim(x);
      putlog x{i}=;
      y = "*" || x{i} || "*";
      putlog "with spaces " y=;
      y = "*" || trim(left(x{i})) || "*";
      putlog "trim(left()) " y=;
      y = "*" || trimn(left(x{i})) || "*";
      putlog "trimn(left()) " y=;
      y = "*" || strip(x{i}) || "*";
      putlog "strip " y=;
      putlog;
   end;
run;
```

SAS Log 1.5: Removing Spaces

```
x1=abc
with spaces y=* abc  *
trim(left()) y=*abc*
trimn(left()) y=*abc*
strip y=*abc*

x2=
with spaces y=*    *
trim(left()) y=* *
trimn(left()) y=**
strip y=**
```

The STRIP() function is great, and you should be using it.

Finding Non-Blank Values

There might be times when you have a set of variables that are either blank or not, and you want to find the one that is not blank. For example, you might have a first-name variable and a nickname variable, and you want to use the nickname unless it is blank. Otherwise, you will use the first name. To do this, you could use IF-THEN-ELSE. However, there is a function that can do it for you: COALESCE(). It is very useful in both PROC SQL and the DATA step.

In PROC SQL it is invaluable when joining tables. If the same column is in multiple tables, you can use the COALESCE() function to get the one that has a non-blank value. In addition, you can set a default if all the values are blank.

The code in Program 1.7 creates two sample data tables that are used in the Program 1.8 example, and Figure 1.2 and Figure 1.3 are listings of those tables.

Program 1.7: Creating Two Tables

```
data table1;
   input x y;
   datalines;
1 1
3 .
5 5
6 6
;
data table2;
   input x y;
   datalines;
1 10
2 20
4 40
7 .
;
```

Figure 1.2: Creating Two Tables—table1

Obs	x	y
1	1	1
2	3	.
3	5	5
4	6	6

Figure 1.3: Creating Two Tables—table2

Obs	x	y
1	1	10
2	2	20
3	4	40
4	7	.

You can join them with PROC SQL and use a CASE statement to get the *x* and *y* values set correctly. That works, but is a lot of work (especially if you can't always remember the CASE statement syntax):

```
proc sql;
   create table newTable as
      select t1.x as t1_x, t2.x as t2_x,
             t1.y as t1_y, t2.y as t2_y,
             case
                when (t1.x ne .) then t1.x
                when (t2.x ne .) then t2.x
                else 0
             end as x,
             case
                when (t1.y ne .) then t1.y
                when (t2.y ne .) then t2.y
                else 0
             end as y
         from table1 as t1
               full join
              table2 as t2
                 on t1.x eq t2.x;
quit;
```

Program 1.8 shows how you can use the COALESCE() function without bothering with the CASE statement, and Figure 1.4 is a listing of the joined data.

Program 1.8: Using COALESCE() in an SQL Join

```
proc sql;
   create table newTable as
      select t1.x as t1_x, t2.x as t2_x,
             t1.y as t1_y, t2.y as t2_y,
             coalesce(t1.x, t2.x, 0) as x,
             coalesce(t1.y, t2.y, 0) as y
         from table1 as t1
               full join
              table2 as t2
                 on t1.x eq t2.x;
quit;
```

Figure 1.4: Using COALESCE() in an SQL Join

Obs	t1_x	t2_x	t1_y	t2_y	x	y
1	1	1	1	10	1	1
2	.	2	.	20	2	20
3	3	.	.	.	3	0
4	.	4	.	40	4	40
5	5	.	5	.	5	5
6	6	.	6	.	6	6
7	.	7	.	.	7	0

The COALESCE() function also works in a DATA step and helps you get rid of some IF statements. Program 1.9 creates a table to use in the example in Program 1.10, and Figure 1.5 is a listing of the table created.

Program 1.9: Creating a Table with Missing Character Values

```
data table;
   input @1 a $1. @3 b $1.;
   datalines;
A B

  C
D
;
```

Figure 1.5: Creating a Table with Missing Character Values

Obs	a	b
1	A	B
2		
3		C
4	D	

Suppose you want to create a new column that is either the value from a or the value from b, whichever is not blank. If they are both blank, then set the value to "Z". You could use an IF-THEN-ELSE statement like this:

```
if (a ne "") then
   newVar = a;
else if (b ne "") then
   newVar = b;
else
   newVar = "Z";
```

Or you can use the COALESCE() function and avoid all that typing, as shown in Program 1.10 and SAS Log 1.6.

Program 1.10: Handling Blank Values with COALESCE()

```
data _null_;
   set table;
   newVar = coalescec(a, b, "Z");
   putlog a= b= newVar=;
run;
```

SAS Log 1.6: Handling Blank Values with COALESCE()

```
a=A b=B newVar=A
a= b= newVar=Z
a= b=C newVar=C
a=D b= newVar=D
```

Tip: If you are coalescing character values in a DATA step, you must use the COALESCEC() function; the COALESCE() function is only for numeric values. However, you can use COALESCE() in PROC SQL for either numeric or character values.

And finally, you can use the COALESCE() function to set a missing value to a constant in a single statement. So the following code can be replaced:

```
if (string eq "") then
   string = "BLANK";
```

Instead, you can do this:

```
string = coalescec(string, "BLANK");
```

I have a friend who likes the coalesce() function so much that he asks prospective SAS programmers to explain it in interviews.

Creating Datetime Values

If you have a SAS date value (stored as the number of days since January 1, 1960) and a SAS time value (the number of seconds since midnight), and you need to create a SAS datetime value (the number of seconds since January 1, 1960), you can use the DHMS() function.

The DHMS() function takes a SAS date value, an hour value (the hour part of a time), a minute value (the minute part of a time), and a second value (the seconds part of a time), and returns a SAS datetime value. Here is the syntax:

```
datetime = dhms(date, hour, minute, second);
```

Program 1.11 and SAS Log 1.7 show that if you only have a date value, DHMS() can also be used to make a SAS datetime value, using zeros for the hour, minute, and second arguments.

Program 1.11: Creating a Datetime Value with No Time Value

```
data x;
   date = "01Jan2016"d;
   datetime = dhms(date, 0, 0, 0);
   putlog date= date9.;
   putlog datetime= datetime16.;
run;
```

SAS Log 1.7: Creating a Datetime Value with No Time Value

```
date=01JAN2016
datetime=01JAN16:00:00:00
```

What about when you have a date and a time? I have often done this to create a datetime value:

```
datetime = dhms(date,
           hour(time),
           minute(time),
           second(time));
```

This code works, but there is some extra effort (both typing and processing) to split out the hour, minute, and second values from the time. However, because a SAS time is stored as the number of seconds since midnight, and because the fourth argument to DHMS() is seconds, you just need to use the time variable as the seconds argument with zeros for the hours and minutes, as you can see in Program 1.12 and SAS Log 1.8:

Program 1.12: Creating a Datetime Value with Date and Time Values

```
data _null_;
   date = "01Jan2016"d;
   time = "13:25"t;
   datetime = dhms(date, 0, 0, time);
   putlog date= date9.;
   putlog time= time8.;
   putlog datetime= datetime16.;
run;
```

SAS Log 1.8: Creating a Datetime Value with Date and Time Values

```
date=01JAN2016
time=13:25:00
datetime=01JAN16:13:25:00
```

Creating Macro Variables

If you work much with macros and macro variables, you have probably used CALL SYMPUT() to create a macro variable from a DATA step value. You might have even used CALL SYMPUTX() to create a local or global macro variable—to ensure that the macro variable is created with the correct macro scope. What you might not realize about CALL SYMPUTX() is that it can make a macro variable out of a numeric DATA step variable, so you don't have to convert the value to a character.

Without using CALL SYMPUTX(), you have probably done something like this:

```
call symput("macroVar", trim(left(put(numVar, 10.1))));
```

And, if you are familiar with the CAT() functions, you can simplify this:

```
call symput("macroVar", cats(numVar));
```

Program 1.13 and SAS Log 1.9 show that with CALL SYMPUTX(), you can simplify the code even more.

Program 1.13: Creating a Macro Variable with CALL SYMPUTX()

```
data _null_;
   numVar = 14.5;
   call symputx("macroVar", numVar);
run;
%put &=macroVar;
```

SAS Log 1.9: Creating a Macro Variable with CALL SYMPUTX()

```
MACROVAR=14.5
```

The CALL SYMPUTX() routine can also help you with character strings. It trims the leading and trailing blanks before assigning the string to the macro variable, so you don't need to use TRIM(LEFT()) or STRIP() to make sure that your macro value isn't padded with blanks.

Finding Words

The SCAN() function is a great tool for splitting a string into chunks using some sort of delimiter. For example, if you have a name such as "Mary Jones", you can use SCAN() to pull off the first word "Mary" and the second word "Jones" to store them in separate variables:

```
firstName = scan("Mary Jones", 1, " ");
lastName = scan("Mary Jones", 2, " ");
```

If you want to get the last word or words from a string, there are lots of ways to do it. You can use a loop and keep looping until you get to the end of the string:

```
count = 0;
lastWord = "";
do until(word eq "");
   count + 1;
   word = scan(longString, count, " ");
   if (word ne "") then
      lastWord = word;
   else
      leave;
end; /* do until - end of string */
```

You can reverse the string, take the first word, and then reverse that word:

```
word = reverse(scan(reverse(strip(longString)), 1, " "));
```

You can use the COUNTW() function to scan the last word:

```
word = scan(longString, countw(longString, " "), " ");
```

And I'm sure you can think of other ways to do this. However, the easiest way is to use the feature of the SCAN() function that allows you to scan from the end of a string. This feature was introduced in SAS 8. All you need to do is use a negative counter; this tells SCAN() to start at the end of the string. So use -1 to get the last word, -2 to get the second-to-last word, and so on. If you know the length of the string and the length of the substrings, then you can just use the SUBSTR() function to pull out the last word. But if you don't have that information, the SCAN() function can handle it for you. Program 1.14 and SAS Log 1.10 show how to get the last and the second-to-last values from a string.

Program 1.14: Finding the Last Word with a Negative Counter

```
data _null_;
   longString = "Word1 Word2 Word3 Word4";
   putlog longString=;

   length wordLast wordSecondLast $10;

   wordLast = scan(longString, -1, " ");
   putlog wordLast=;

   wordSecondLast = scan(longString, -2, " ");
   putlog wordSecondLast=;
run;
```

SAS Log 1.10: Finding the Last Word with a Negative Counter

```
longString=Word1 Word2 Word3 Word4
wordLast=Word4
wordSecondLast=Word3
```

The SCAN() function is incredibly useful, and has lots of parameters that you might have missed (as I did).

Counting Words

If you have a string that consists of values that are delimited with some character, you might want to know how many words are in that string, or you might want to do different processing based on the number of strings. For example, if you have a name string such as "Tom Jones" or "Sally Jo Smith", you need to check the number of words to determine whether there is a middle name or not before you separate the parts of the name.

The COUNTW() function can tell you how many words are in a string. A word is defined as a substring that is bounded by delimiters and the beginning and end of the string. COUNTW() saves you having to scan words from a string until you get a blank value, signaling that you have reached the end of the string, like this:

```
count = 0;
do until(word eq "");
   count + 1;
   word = scan(longString, count, " ");
   if (word ne "") then
      putlog count= word=;
end; /* do until - end of longString */
```

The example in Program 1.15 and SAS Log 1.11 shows how you can use the COUNTW() function to simplify your code.

Program 1.15: Using COUNTW() to Get the Number of Words

```
data _null_;
   longString = "Word1 Word2 Word3 Word4";
   length word $10;
   do count = 1 to countw(longString, " ");
      word = scan(longString, count, " ");
      putlog count= word=;
   end; /* do count - loop through words */
run;
```

SAS Log 1.11: Using COUNTW() to Get the Number of Words

```
count=1 word=Word1
count=2 word=Word2
count=3 word=Word3
count=4 word=Word4
```

The code in Program 1.16 and SAS Log 1.12 shows how to use COUNTW to split apart a name that might include a middle name and might not, you can use countw() to check the number of words to see whether there are 2 words (first and last names only) or 3 words (first, middle, and last names).

Program 1.16: Using COUNTW to Split a Name

```
data _null_;
   input name $20.;
   putlog name=;

   length firstName middleName lastName $20;
   firstName = scan(name, 1, " ");
   if (countw(name) eq 3) then
      middleName = scan(name, 2, " ");
   lastName = scan(name, countw(name), " ");

   putlog firstName= middleName= lastName=;
   datalines;
Mary Jones
Tom A. Smith
;
```

SAS Log 1.12: Using COUNTW to Split a Name

```
name=Mary Jones
firstName=Mary middleName= lastName=Jones
name=Tom A. Smith
firstName=Tom middleName=A. lastName=Smith
```

The COUNTW() function has plenty of arguments that enable you to specify the correct delimiters.

Replacing Substrings

The TRANSTRN() function and its sister (brother?), tranwrd() function, enable you to replace part of a string with another string. The primary difference between the two is that TRANSTRN() lets you remove part of a string without leaving a blank in its place.

> **Tip**: The TRANWRD() function was introduced in SAS 9.1, and the TRANSTRN() function followed in SAS 9.2. I got into the habit of using TRANWRD(), so I tend to use it most of time. If you haven't gotten into this habit, then ignore TRANWRD() and use TRANSTRN() instead; it has all the same functionality as TRANWRD(), plus the ability to remove a string without leaving a blank.

You can achieve the same results by using functions such as INDEX(), SUBSTR(), and TRANSLATE(), but TRANSTRN() is so much easier. For example, without using TRANSTRN() you could use INDEX() and SUBSTR() and a DO loop to replace all blanks with "???":

```
newString = string;
do until (i eq 0);
   i = index(strip(newString), " ");
   if (i gt 0) then
    newString = substr(newString, 1, i-1) || "???" ||
                substr(newString, i+1);
end;
```

And you can remove all instances of a string (in this case, "???") like this:

```
do until (i eq 0);
   i = index(strip(newString), "???");
   if (i gt 0) then
    newString = substr(newString, 1, i-1) ||
                substr(newString, i+3);
end;
```

Program 1.17 and SAS Log 1.13 show that you can use TRANSTRN() to simplify both of these tasks considerably.

Program 1.17: Replacing Parts of a String Using TRANSTRN()

```
data _null_;
   string = "This is a test";
   putlog string=;

   /* replace all blanks with "???" */
   newString = transtrn(string, " ", "???");
   putlog newString=;

   /* remove all "???" */
   newString = transtrn(newString, "???", trimn(""));
   putlog newString=;
run;
```

SAS Log 1.13: Replacing Parts of a String Using TRANSTRN()

```
string=This is a test
newString=This???is???a???test
newString=Thisisatest
```

Tip: When you want to remove part of a string without leaving a blank space in its place, use TRIMN("") instead of just "" as the replacement value in TRANSTRN(). This replaces the first string with a zero-length string.

Warning: If you want to use a variable instead of a quoted value for the second or third arguments in TRANSTRN() or TRANWRD(), you probably need to use STRIP() to remove leading and trailing blanks.

If you use an unstripped variable as the target (the second argument), the function looks for the value with all the padding on the end.

And if you use an unstripped variable as the replacement value (the third argument), the function adds all the blanks at the end of the value when it does the replace.

The TRANWRD() and TRANSTRN() functions are incredibly useful when you are cleaning and transforming data.

Using %SYSFUNC() to Run DATA Step Functions

If you haven't used the %SYSFUNC() macro function, you have been missing out on access to most of the SAS DATA step functions from outside the DATA step. With %SYSFUNC() you can run a DATA step function outside a DATA step.

Program 1.18 and SAS Log 1.14 show a simple example that uses %SYSFUNC() with the NOTDIGIT() function to check whether the value in a macro variable is a digit or not (0 indicates that it is a digit, and a number indicates the first position in the string of a non-digit).

Program 1.18: Using %SYSFUNC() with NOTDIGIT()

```
%let string = 1A2;
%let rc = %sysfunc(notDigit(&string));
%put &=rc;
%let string = 12;
%let rc = %sysfunc(notDigit(&string));
%put &=rc;
```

SAS Log 1.14: Using %SYSFUNC() with NOTDIGIT()

```
1   %let string = 1A2;
2   %let rc = %sysfunc(notDigit(&string));
3   %put &=rc;
RC=2
4   %let string = 12;
5   %let rc = %sysfunc(notDigit(&string));
6   %put &=rc;
RC=0
```

If you want to put a date value into a macro variable, you can save the numeric value of the date (which is the number of days since 01JAN1960) or you can save the formatted value as you can see in Program 1.19 and SAS Log 1.15. Note that you can apply a format to the value as the second parameter of %SYSFUNC().

Program 1.19: Using %SYSFUNC() with Dates

```
%let date = %sysfunc(today());
%put &=date;
%let date = %sysfunc(today(), date9.);
%put &=date;
```

SAS Log 1.15: Using %SYSFUNC() with Dates

```
1   %let date = %sysfunc(today());
2   %put &=date;
DATE=20877
3   %let date = %sysfunc(today(), date9.);
4   %put &=date;
DATE=27FEB2017
```

In Program 1.20 and SAS Log 1.16, you can see that %SYSFUNC() can be used to do lots of other work as well, including reading data tables.

Program 1.20: Using %SYSFUNC() to Read a Table

```
%macro readData;
   %let did = %sysfunc(open(sashelp.class (where=(sex='F'))));
   %if (&did eq 0) %then
   %do;
      %put %sysfunc(sysmsg());
      %return;
   %end;
   %do i = 1 %to %sysfunc(attrn(&did, NLOBSF));
      %let rc = %sysfunc(fetchobs(&did, &i));
      %let name = %sysfunc(getvarc(&did,
                           %sysfunc(varnum(&did, name))));
      %let age = %sysfunc(getvarn(&did,
                           %sysfunc(varnum(&did, age))));
      %put &=name &=age;
   %end;
   %let did = %sysfunc(close(&did));
%mend;
%readData;
```

SAS Log 1.16: Using %SYSFUNC() to Read a Table

```
NAME=Alice AGE=13
NAME=Barbara AGE=13
NAME=Carol AGE=14
NAME=Jane AGE=12
NAME=Janet AGE=15
NAME=Joyce AGE=11
NAME=Judy AGE=14
NAME=Louise AGE=12
NAME=Mary AGE=15
```

When using %SYSFUNC(), do not enclose the function parameters in quotation marks even if you would normally do that in a DATA step. For example, in a DATA step, you would use this syntax:

```
string = transtrn(string, "a", "A");
```

When using %SYSFUNC() outside a DATA step, the syntax would be as follows:

```
%let string = %sysfunc(transtrn(&string, a, A));
```

Another thing to be aware of is that you have to use a separate %SYSFUNC() for each function you call:

```
%let name = %sysfunc(getvarc(&did,
                     %sysfunc(varname(&did, 3))));
```

There are some functions that cannot be used with %SYSFUNC():

ALLCOMB()	LEXCOMB()
ALLPERM()	LEXCOMBI()
DIF()	LEXPERK()
DIM()	LEXPERM()
HBOUND()	MISSING()
IORCMSG()	PUT()
INPUT()	RESOLVE()
LAG()	SYMGET()
LBOUND()	variable information functions—VNAME(), VLABEL(), and the like

Instead of PUT() and INPUT(), you can use PUTN(), putc(), inputn(), and INPUTC().

About This Book

What Does This Book Cover?

This book does not teach a novice how to program with SAS. Its purpose is to provide SAS programmers with better tools for accomplishing a variety of tasks using DATA steps, macros, the SQL procedure, and other procedures.

It covers data manipulation, tricks for writing better programs, utilities to simplify application development, and a variety of techniques, including using recursion, using lookup tables, and reading metadata.

Is This Book for You?

The expected audience for this book is SAS programmers with some—or a lot of—experience in writing SAS code. You should have some experience with the DATA step (basic concepts and the use of functions) and with the macro facility (macro variables and simple macro programming). In addition, a basic understanding of PROC SQL and other standard procedures is useful.

What Should You Know about the Examples?

This book includes examples that you can run for yourself to gain hands-on experience with SAS. All of the examples are self-contained. Most of the data that is used is from the standard sashelp sample tables (for example, sashelp.class and sashelp.cars). A few examples create their own data tables.

Software Used to Develop the Book's Content

Base SAS 9.4 was used to develop the content for this book. Some of the code works in any version of SAS, some only works in SAS 9, and some only works in SAS 9.4. In addition, there is one example that requires access to the SAS Metadata Repository, formerly known as the *SAS Open Metadata Repository*, so the SAS Metadata Server must be installed to use these examples.

The code was primarily developed on a Windows 7 platform and runs on any operating system that SAS runs on. However there are some examples that are written specifically for Windows and/or Linux and would need to be modified to run on a different operating system.

Example Code and Data

You can access the example code for this book by linking to its author page at https://support.sas.com/authors.

All the data used in the example programs is available in the standard sashelp library available to all SAS users or is created by the example program.

We Want to Hear from You

SAS Press books are written *by* SAS Users *for* SAS Users. We welcome your participation in their development and your feedback on SAS Press books that you are using. Please visit sas.com/books to do the following:

- sign up to review a book
- recommend a topic
- request information on how to become a SAS Press author
- provide feedback on a book

Do you have questions about a SAS Press book that you are reading? Contact the author through saspress@sas.com or https://support.sas.com/author_feedback.

SAS has many resources to help you find answers and expand your knowledge. If you need additional help, see our list of resources: sas.com/books.

About The Author

Martha Messineo is a principal software developer in the SAS Solutions OnDemand division at SAS. She has more than 30 years of SAS experience, most of which she has spent writing SAS programs in a variety of positions, including those in Technical Support, Professional Services, Management Information Systems, SAS Solutions OnDemand, and Research and Development. In her Research and Development role, she worked with SAS Data Integration Studio, IT Resource Management, and SAS Asset Performance Analytics. Before coming to SAS, she worked for MetLife Insurance in the Capacity Planning and Performance Tuning division, where she used SAS to analyze mainframe performance data. She is a SAS Certified Base Programmer for SAS9, a SAS Certified Advanced Programmer for SAS9, and a Data Integration developer for SAS9.

Learn more about this author by visiting her author page at http://support.sas.com/Messineo. There you can download free book excerpts, access example code and data, read the latest reviews, get updates, and more.

Ready to take your SAS® and JMP® skills up a notch?

Be among the first to know about new books, special events, and exclusive discounts.
support.sas.com/newbooks

Share your expertise. Write a book with SAS.
support.sas.com/publish

sas.com/books
for additional books and resources.

sas
THE POWER TO KNOW®

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.
Other brand and product names are trademarks of their respective companies. © 2017 SAS Institute Inc. All rights reserved. M1588358 US.0217