Paper SA06-2014

Working with Character Data

Andrew T. Kuligowski, HSN
Swati Agarwal, Optum

ABSTRACT

The DATA Step allows one to read, write, and manipulate many types of data. As data evolves to a more free-form state, SAS's ability to handle character data becomes increasingly important.

This presentation will address character data from multiple vantage points. For example, what is the "default" length of a character string, and why does it appear to change under different circumstances? What sort of formatting is available to us for character data? How can we examine and manipulate character data?

This paper will aim at a beginning to intermediate audience, and will stress an introduction to the numerous character functions available in SAS®, including the basic LENGTH, and SUBSTR, plus many others.

WHAT IS "CHARACTER DATA"

The folks at Microsoft describe it succinctly: "Character data consists of any combination of letters, symbols, and numeric characters." Early computer users dealt with encoding methods such as ASCII and EBCDIC, in which characters are converted to a one-byte numeric representation. This allows the user and the machine to "communicate", as there is a representation of every English-language character in both upper and lower case, the digits 0 through 9, and select special characters – basically, whatever could be found on a standard typewriter – plus a few special non-printable characters representing internal functionality like Carriage Return (CR) and Line Feed (LF).

The limitations of these approaches should be immediately obvious to individuals used to working in the world-wide community. How does one type letters found in non-English languages – the ñ and ¿ in Spanish, for example? The approach even proved insufficient for English: take, for example, the curved quotation marks angled inward to indicate the beginning and end of a quote?

Nowadays, Unicode provides a common encoding scheme which crosses countries, continents, and languages. While it counters the weakness of a limited character set inherent in other methods, it also counters their strength, in that it takes more than 1 byte to uniquely identify most characters.

HOW LONG IS MY CHARACTER VARIABLE?

Of course, in most situations, you want to store more than a single character. In its early days, SAS used to limit the user to a maximum of 200 characters per value. Of course, you could store more than 200 characters in an observation, but in order to do so, it was necessary to split your value into multiple variables. Nowadays, that maximum length of a variable has been increased to 32,767 bytes.

The length of a variable is determined from the variables' first usage. If the variable's first reference finds it hardcoded to a string, the length of that string will be used to determine the variable's length. In other cases, the length of the variable(s) used in its assignment will determine its length. The safest way to assign a length to a variable is with the LENGTH statement. However, the LENGTH statement must contain the first reference to the variable in the routine. Otherwise, SAS will produce the following notification:

```
WARNING: Length of character variable y has already been set. Use the
         LENGTH statement as the very first statement in the DATA STEP
         to declare the length of a character variable.
```

Examples can be found in **TABLE 1**.

```
data temp;
   length w $ 2  y $ 40.;
   retain w "";
   x = "abcdefghijklmnopqrstuvwxyz";
   y = "";
   z = "1234567890";
   output;
```

*LENGTH statement: W will have a length of 2, and Y will have a length of 40.
First reference: X will have a length of 26, and Z will have a length of 10. (The length of Y will not change; it has already been set.)*

```
   y = "----+----1----+----2----+----3----+----4----+----5";
   a1 = substr(x,14);
   a2 = substr(x,14,13);
```

(The length of Y will not change; it has already been set. However, the assigned value will be truncated at 40 bytes.)

First reference: A1 and A2 will have a length of 26, based on the length of the variable they were created from. (SAS does not factor in the length that we're not using the entire length of the source variable when determining the length of the new variable.)

```
   a3 = x || y ;
   a4 = substr( x || y, 25, 1 ) || substr( x || y, 26, 1 );
   output;
run;
```

First reference: A3 will have a length of 66, based on the cumulative length of the variables used to create it.) A4 will have a length of 132, also based on the cumulative length of the variables used to create it – even though we are only effectively using 2 bytes..)

Table "1" : Default length settings

The "first usage" rule also applies when combining two or more datasets, such as with a SET or MERGE statement. The first reference to a variable will define its length. Let us take dataset X, which contains a variable STRING with a length of 10, and dataset Y, which also contains a variable STRING with a length of only 4. Combining the two datasets with the statement SET X Y; will result in STRING being defined with a length of 10. All 4-character strings from dataset Y will be blank-padded to a length of 10. On the other hand, the statement SET Y X; will result in STRING only having a length of 4. This will result in the values of STRING contributed from dataset X having their last 6 characters truncated. SAS does not note the blank padding in the SASLOG, but will caution about the latter case's possible loss of data with the following message:

```
WARNING: Multiple lengths were specified for the variable String by
         input data set(s). This may cause truncation of data.
```

MANIPULATING CHARACTER DATA

Often, creating or acquiring character data is all you need to do. "Get it" / "Create it", then "store it" and/or "report it". There are other instances where it is necessary to perform some evaluation and transformation on character data. SAS provides a number of useful functions to assist in this effort.

Concatenation and Truncation

Some readers may recall that in the original Karate Kid movie, Mr. Miyagi taught Daniel to block before he taught him to strike. It is similarly necessary to talk about truncation – the removal of one or more characters from a string – before we can properly discuss concatenation, which is the addition of characters to a string or the combining of 2 or more strings into a new string. As in the movie, the reasons will be

TRIM(<*string*>) and TRIMN(<*string*>) both remove trailing blanks from a character string. TRIM will leave a single blank if the string is missing, while TRIMN will return a string with a length of zero. (Neither should be confused with TRUNCATE, which is a numeric function.) STRIP(<*string*>) performs a parallel task, removing leading blanks from a character string.

COMPRESS(<*string*>), in its simplest form, combines the functionality of both STRIP and TRIM, removing both leading and trailing blanks from a string by default. However – and this may be perceived as either a bonus or a detriment depending on the particular circumstance, the COMPRESS function removes ALL blanks from the string, including any embedded blanks in the middle of the string.

Observant readers will note that the previous paragraph clearly uses words like “simplest” and “default”. There is an optional parameter and a set of modifiers that will increase the ability of the COMPRESS function. When dealing with COMPRESS(<*string*>, <*char-list*>, <*modifiers*>), the parameter <*char-list*> will change the character to be removed from the string from a blank to, well, to anything. And “anything” is not limited to a single character – an entire set of characters including a blank could be provided and COMPRESS-ed out of your original string. The various modifiers add categories of characters to the list of characters to be compressed out, such as lowercase letters (“L” or “l”), digits (“D” or “d”), and others. One exception is the “k” – this modifier causes COMPRESS to only KEEP the characters in the string and COMPRESS-es out all characters NOT found in the string!

Traditionally, you can concatenate (join) two strings using the concatenation operator (either || or !!). This has one built-in weakness – leading and trailing blanks are considered to be part of the component strings being concatenated. This means that 3 variables with a length of 20 bytes each – 1 character and 19 trailing blanks, to take an extreme example – will form a 60 byte string if concatenated with ||. This will consist of the 1 character followed by 19 blanks, then 1 character followed by another 19 blanks, and ending with 1 character followed by 19 blanks. (This is why we discussed truncation before concatenation!)

There are several new functions and CALL routines with SAS® 9 that can be used to alleviate the additional processing and potential errors that accompany these trailing blanks.. The advantage of these new functions – known collectively as the CAT functions – is that they can automatically strip off leading and trailing blanks, and can insert separation characters for you.

The **CATS function** strips leading and trailing blanks before joining two or more strings:

```
CATS(string1,string2,<stringn>);
```

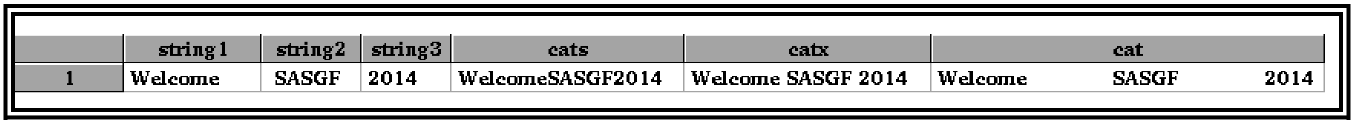
The **CATX function** works the same as the CATS function but allows you to specify one or more separation characters to insert between the strings. The syntax for these two functions is:

```
CATX(separator,string1,string2,<stringn>);
```

TABLE 2 contains an example of the use of these two CAT functions, while **TABLE 3** shows the output of the example.

```
data join_up;
  length string1-string3 $ 20;
  length cats $ 17  catx $ 20;
  string1 = 'Welcome ';
  string2 = ' SASGF ';
  string3 = '2014';
  cat  = string1 || string2 || string3;
  cats = cats(string1,string2,string3);
  catx = catx(' ',string1,string2,string3);
run;
```

Table "2" : CATS and CATX functions



	string1	string2	string3	cats	catx	cat
1	Welcome	SASGF	2014	WelcomeSASGF2014	Welcome SASGF 2014	Welcome SASGF 2014

Table "3" : CATS and CATX output

As discussed earlier, the LENGTH statement specifies the number of characters to be allotted to a string. Without the LENGTH statement in this program, the length of the variables CATS and CATX would be 200, which is the default length for the CAT functions. The length of CAT, which has no associated LENGTH statement and is created using a concatenation operator, is set to the total length of its component variables. In this case, our variable is made from 3 variables of 20 bytes each, for a total of 60 bytes.

There are 3 other CAT functions: CAT, CATQ, and CATT. CAT performs a simple concatenation without stripping leading or trailing blanks, similar to what you could get using the concatenation operator || without the TRIM function, both of which was discussed earlier. CATQ is similar to CATX, but will add quotation marks to strings that contain the specified delimiter. (This is useful in the event one wants to create a Comma Separated Value, or CSV file, in which some of the strings actually contain commas.). CATT is also similar to CATX, except the 2nd T stands for “Trailing”; only trailing blanks are trimmed from the resulting value while leading blanks are left alone. Space and time limitations prevent a further examination of these functions.

Due to space limitations, we will not include substrings in this edition of the presentation. Readers are encouraged to look in the online manual for functions such as SUBSTR, SUBSTRN, and CHAR.

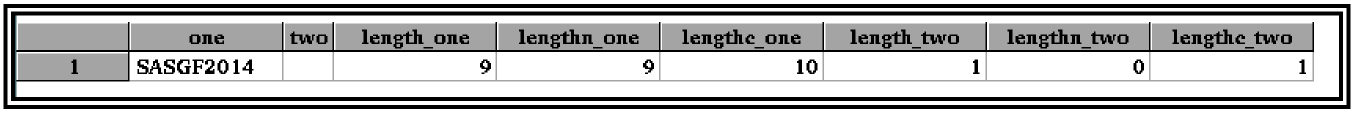
The LENGTH, LENGTHN, and LENGTHC Functions

There are several LENGTH functions available in SAS which have different purposes. The LENGTHC function (V9) returns the storage length of character variables. The other two functions, LENGTH and LENGTHN both return the length of a character variable *not counting trailing blanks*. The only difference between LENGTH and LENGTHN is that LENGTHN returns a 0 for a null string while LENGTH returns a 1. This is demonstrated in **TABLE 4**, with the output displayed in **TABLE 5**.

```
data how_long;
  one = 'SASGF2014 ';
  two = ' '; /* character missing value */
  length_one = length(one);
  lengthn_one = lengthn(one);
  lengthc_one = lengthc(one);

  length_two = length(two);
  lengthn_two = lengthn(two);
  lengthc_two = lengthc(two);
run;
```

Table "4" : UPCASE, LOWCASE, and PROPCASE functions



	one	two	length_one	lengthn_one	lengthc_one	length_two	lengthn_two	lengthc_two
1	SASGF2014		9	9	10	1	0	1

Table "5" : UPCASE, LOWCASE, and PROPCASE output

Manipulation – UPCASE and related functions

Let us look at a few ways to manipulate the characters within a string, rather than simply dealing with matters of its length.

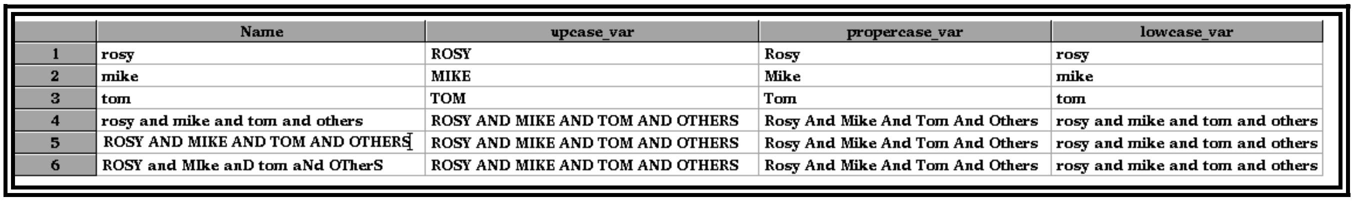
The two companion functions UPCASE and LOWCASE do just what you would expect. These two functions are especially useful when data entry clerks are careless and a mixture of upper and lower cases values are entered for the same variable.

In order to convert all characters in a sting to uppercase, use the UPCASE function. Conversely, the LOWCASE function will convert them to lower case. PROPCASE will capitalize the first letter of word in a string, leaving – or converting – the others to lower case. See **TABLE 6** for an example of each function, and **TABLE 7** for the output of that example.

```
data ds_1;
  length Name $ 80. ;
  input Name $ & ;
datalines;
rosy
mike
tom
rosy and mike and tom and others
ROSY AND MIKE AND TOM AND OTHERS
ROSY and MIke anD tom aNd OTherS
;

data ds_2;
  set ds_1;
  *convert it to other cases;
  upcase_var     = upcase(Name);
  propercase_var = propcase(Name);
  lowcase_var    = lowcase(upcase_var);
run;
```

Table "6" : UPCASE, LOWCASE, and PROPCASE functions



	Name	upcase_var	propercase_var	lowcase_var
1	rosy	ROSY	Rosy	rosy
2	mike	MIKE	Mike	mike
3	tom	TOM	Tom	tom
4	rosy and mike and tom and others	ROSY AND MIKE AND TOM AND OTHERS	Rosy And Mike And Tom And Others	rosy and mike and tom and others
5	ROSY AND MIKE AND TOM AND OTHERS	ROSY AND MIKE AND TOM AND OTHERS	Rosy And Mike And Tom And Others	rosy and mike and tom and others
6	ROSY and MIke anD tom aNd OTherS	ROSY AND MIKE AND TOM AND OTHERS	Rosy And Mike And Tom And Others	rosy and mike and tom and others

Table "7" : UPCASE, LOWCASE, and PROPCASE output

Searching (e.g. INDEX, VERIFY, ANYALPHA)

There is an entire family of routines built around searching a character string for particular components. Functions such as INDEX, FIND, VERIFY, and the entire set of ANY- functions (ANYALPHA, for example) are set up to locate particular characters or strings in a target string. A more complete analysis of these types of functions will be provided in a future edition of this presentation.

Perl Regular Expressions

Regular Expressions facilitate a search for patterns in character data. Perl regular expressions were introduced into SAS with Version 9, Their syntax and coding are not trivial, and can appear confusing to the neophyte SAS users. (Also to the experience SAS user.) This complexity can intimidate the user into avoiding them and sticking with the old, familiar SAS functions that they've been using for years. However, this complexity is also their strength – you can perform much more complex tasks with Perl regular expressions than is possible with parallel SAS functions – at least without having to write more complex data step / macro statements surrounding them.

Liberally adapting an example from Cody (2008), It is relatively easy to code a FIND statement to look for a particular character or set of characters within a larger string. However, if the request was to look for a set of characters consisting of 3 characters in which all 3 were numeric and the last was a zero (0), you could use

```
IF _N_ = 1 THEN PATTERN_NEEDLE = PRXPARSE( "/\d\d0/" );
```

to define the search at the beginning of the routine, and the function

```
PRXMATCH( PATTERN_NEEDLE, STRING_HAYSTACK )
```

to actually search for the needle in the haystack.

Given time and space constraints, the reader is encouraged to refer to works documented in the REFERENCES section for more detailed information about Perl Regular Expressions.

CHARACTER FORMATS

SAS provides a number of Character INFORMATs and FORMATs that can be used when data is read in for the first time, and when data is output to a subsequent dataset, report, or whatever destination(s) are preferred by the user. One of the nice features of SAS is that it is acknowledged that the standard set may be insufficient for the users' needs. As such, through PROC FORMAT, it is possible to create customized formats – both standardized and "on the fly" – in each SAS site. It is also possible, using the INPUT and PUT functions, to use these formats to perform data transformations within a DATA step.

Space and time limitations prevent a detailed coverage of this material; as with Perl Regular Expressions, full papers and presentations are written purely on PROC FORMAT alone. Some additional material will be included in the References section for those that wish to further examine this powerful functionality.

SOME THINGS TO REMEMBER

There are a few things to keep in mind when working with character data in SAS:

- The length of the variable is determined by its first occurrence in your routine.
- Blank padding should be considered part of your value when performing concatenation.
- (This hasn't been specifically mentioned yet, but …) Manipulation of character data occurs independently of the length of the resulting variable. You can perform some marvelous things within your DATA step that never get written to your output dataset.

As such, it is possible to perform coding tricks that will thoroughly confuse people – starting with the person who entered them in the first place.

As an aside, since the topic is "character data": It should be noted that ALL macro variables contain character data. It is possible to perform mathematical processes on values that contain numbers, but it is necessary to advise SAS via macro function that you are doing so.

ENHANCEMENTS IN RECENT RELEASES

People who are new to using SAS get to learn on the newest version of the language – or at least a reasonably current version. Veteran users who "know how to do something" may not keep current with new developments. As a nod to those individuals, let us take a quick look at what has been introduced in the past couple of releases of SAS in the area of character data.

SAS 9.3 did not bring a lot of changes and enhancements to the ability to process and analyze character data. It did allow enhanced use of characters in variable and dataset names. Enhancements such as VALIDVARNAME=ANY (or EXTEND) permit special and national characters in variable names, and the new function MVALID, will show whether a given character string can be used as a SAS member name. The only new feature that actually involved processing of character data is the new automatic macro variable SYSSIZEOFUNICODE, which contains the length in bytes of a Unicode character in the current session.

Licensed sites that upgrade to SAS 9.4 will find a couple of new features to existing character functions, and a few additional functions to deal with character data. PUTC and PUTN now allow justification of the results, and SCAN now defaults the length of the resulting character variable to the length of the first string processed – unless the variable's length had been defined prior to the function being executed.

There are a few new functions involving character data with Version 9.4. TYPEOF will indicate whether an argument is character or numeric, but it can only be used in WHERE clauses and the Graphic Template Language (GTL). FCOPY will copy a record from one Fileref to another fileref, facilitating the movement of data from one file to another. DOSUBL will allow the exchange of macro variables between SAS and the calling environment. (Note that this is refers to the operating system's macro variable, not SAS's.) And, for cryptologists, has tag fans, and others, SHA256 will return the result of a message digest of a specified string.

CONCLUSION

Given the broad nature of the topic, it could take an entire book to cover the concept of character data – it's definition and use, its exploration and manipulation, its input and output. That sort of coverage is simply not possible in this format. It is the authors' hope that this material provides sufficient material to provide a cursory explanation to the reader regarding simple use and exploration of character data, and to whet their appetite to explore the topic further on their own.

BIBLIOGRAPHY / REFERENCES

Bilenas, Jonas V.. (2008) "I Can Do That With PROC FORMAT". *Proceedings of the SAS® Global Forum 2008 Conference*. Cary, NC: SAS Institute Inc. (Also available at*: http://www2.sas.com/proceedings/forum2008/174-2008.pdf)*

Borowiak, Kenneth W (2007) "Perl Regular Expressions 102"..*Proceedings of the SAS® Global Forum 2007 Conference*. Cary, NC: SAS Institute Inc. (Also available at*: http://www2.sas.com/proceedings/forum2007/135-2007.pdf)*

Cassell. David L. (2007) "The Basics of the PRX Functions". *Proceedings of the SAS® Global Forum 2007 Conference*. Cary, NC: SAS Institute Inc. (Also available at: *http://www2.sas.com/proceedings/forum2007/223-2007.pdf)*

Cody, Ron. (2004) "An Introduction to Perl Regular Expressions in SAS 9". *Proceedings of the Twenty-Ninth Annual SAS® Users Group International Conference*. Cary, NC: SAS Institute, Inc. (Also available at: *http://www2.sas.com/proceedings/sugi29/265-29.pdf*)

Cody, Ron. (2010) *SAS Functions by Example, Second Edition*. Cary, NC: SAS Institute, Inc.

Hemedinger, Chris. (2014) "A fresh helping of hash: the SHA256 function in SAS 9.4m1". *The SAS Dummy: A SAS® blog for the rest of us. http://blogs.sas.com/content/sasdummy/2014/01/18/sha256-function-sas94/*

Kuligowski, Andrew T. (2008) "Using SAS® to Parse External Data". *Proceedings of the SAS® Global Forum 2008 Conference*. Cary, NC: SAS Institute Inc. (Also available at*: http://www2.sas.com/proceedings/forum2008/190-2008.pdf)*

Microsoft. (2008) "Character Data". Microsoft SQL Server Support Site. *http://technet.microsoft.com/en-us/library/aa933107(v=sql.80).aspx*

Patton, Nancy K. (1998) "IN & OUT of CNTL with PROC FORMAT" Proceedings *of the Twenty-Third Annual SAS® Users Group International Conference.* Cary, NC: SAS Institute Inc. (Also available at: *http://www2.sas.com/proceedings/sugi23/Coders/p68.pdf*)

SAS Institute, Inc. (2012). *SAS® 9.3 Functions and CALL Routines: Reference,.* Cary, NC: SAS Institute, Inc.

SAS Institute, Inc. (2013). *SAS® 9.4 Functions and CALL Routines: Reference, Second Edition.* Cary, NC: SAS Institute, Inc.

SAS Institute, Inc. (2012). *SAS® 9.3 Statements: Reference,.* Cary, NC: SAS Institute, Inc.

SAS Institute Inc. (2001). *Step-by-Step Programming with Base SAS® Software*. Cary, NC: SAS Institute Inc. (Also available at: *http://support.sas.com/documentation/cdl/en/basess/58133/PDF/default/basess.pdf*)

SAS Institute, Inc. (2012). *What's New in SAS 9.3.* Cary, NC: SAS Institute, Inc.

SAS Institute, Inc. (2013). *What's New in SAS 9.4.* Cary, NC: SAS Institute, Inc.

Tabladillo, Mark (2012) "Regular Expressions in SAS® Enterprise Guide®". *Proceedings of the SAS® Global Forum 2012 Conference*. Cary, NC: SAS Institute Inc. (Also available at: http://support.sas.com/resources/papers/proceedings12/299-2012.pdf)

Unicode Consortium. (2014). *What is Unicode?*. http://www.unicode.org/standard/WhatIsUnicode.html

ACKNOWLEDGEMENTS / FOR FURTHER INFORMATION

The authors would like to thank the conference planners of SAS Global Forum 2014 for their willingness to see the material in this paper presented at their conference by the two of us. A special thank-you to Lex Jansen, for providing the invaluable resource www.lexjansen.com – this web site allows a quick search of virtually all SAS-related Proceedings by conference, by keyword, and by author.

The authors can be contacted via e-mail at:
KuligowskiConference@gmail.com
swati_agarwal@optum.com