

Are Strings Tying You in Knots?

Deb Cassidy

Cardinal Distribution, Dublin, OH

ABSTRACT

If the answer to the title questions is "YE", then this presentation is for you. You will learn how to make sure you get the entire string "YES" when creating a new variable. But that is just the beginning of some of the fun you can have with character data. This presentation will walk through examples of cleaning up your character data before you can use procedures such as PROC FREQ. Some of the functions to be covered are SCAN, TRIM, LEFT, INDEX, COMPRESS, COMPBL, DEQUOTE and TRANWRD. This presentation uses BASE SAS® and is aimed at beginning users although all users may learn a new, helpful function.

PROBLEM

You have been given a dataset with information about some of the SSU attendees. You have been asked to supply the following summary information.

- # of people in each region
- # of people interested in each subject area
- list of people sorted by last name
- verify ZIPCODE is a valid United States code

The requirements seem easy and you think you will just need the following PROCs: FREQ, SORT, and PRINT. Since you were not given the variable names, you first do a PROC CONTENTS and a PROC PRINT on the first 5 observations. You discover a few complications.

EXAMPLE DATA

The results of PROC CONTENTS and PROC PRINT are shown at the end of the paper. PROC CONTENTS reveals that the data does not even have the fields ZIPCODE, region or last name. At least it does have a variable for interest. When you look at a few records as shown in the PROC PRINT, you see the missing information is really there but not in the form you need. (Please note that the data are fictitious and any resemblance to real attendees is entirely a figment of the author's imagination.)

THE REAL PROBLEM

Now that you have seen the data, you know you really need to do several things before you can run your PROCs.

- Create region from state after state is extracted from ADD2
- Separate INTEREST into only one interest per record
- Separate first and last names from NAME
- Separate ZIPCODE from ADD2

CREATING NEW VARIABLES

The easiest task is to create region from state. (In real life, you have to extract state from ADD2 first but in a tutorial the presenter gets to rearrange things.)

There are several ways to do this. In fact, you could actually use PROC FORMAT and avoid the creation of the new variable region. However, we are going to assume you have a reason for adding the new variable. There are also cases where the logic is more complicated and an IF/THEN or CASE logic solution is the only one feasible. The code would be similar to the following sample section.

```
DATA STATE_REG;
  SET ORIGINAL;
  IF STATE IN ('CA', 'OR', ...) THEN
    REGION='WEST';
  ELSE IF STATE IN ('FL', 'TN', 'NC' ...) THEN
    REGION='SOUTH';
  ELSE IF STATE IN ('OH' ...) THEN
    REGION='MIDWEST';
  ... REST OF STATES...
  ELSE REGION='INVALID';
RUN;
```

```
PROC FREQ DATA=STATE_REG;
  TABLES REGION / NOCUM NOPERCENT;
RUN;
```

The results give you:

<u>REGION</u>	<u>FREQUENCY</u>
INVA	1
MIDW	1
SOUT	2
WEST	1

TUTORIALS

While the counts are right, you notice the regions are truncated at 4 characters. You have heard that PROC FREQ had a limitation but upon further investigation you discover that it used to be limited to the first 16 characters but this limitation no longer exists. So what happened? Run another PROC CONTENTS and you will see the REGION variable was created as character variable with a length of 4. SAS determines the length based on the first time it sees the variable. Since your first value was "WEST", the variable was created with a length of 4. One solution would be to put the longest value first. Unfortunately the longest value is "INVALID" but it always needs to be last. Another choice would be to pad your first value with blanks - "WEST " would solve the problem. This works but is very susceptible to future problems if you need to add values. The better alternative is to use a length statement.

LENGTH REGION \$7.;

...put IF statements AFTER length statement

Your new results are:

<u>REGION</u>	<u>FREQUENCY</u>
INVALID	1
MIDWEST	1
SOUTH	2
WEST	1

Of course, someone will want "INVALID" to appear at the end. They might also want the other values to appear in a specific order. PROC FREQ does have several options that will control order but unfortunately there is not one for "how I typed them." One alternative is to use lead blanks since blanks will sort first. However, that could get quite confusing and would look rather strange. This time PROC FORMAT is probably the best way to solve the problem. Instead of creating the region as a character variable you will want to create it as a numeric variable with the regions in the order that you want to see in your output. PROC FORMAT is used to show the complete region name for the numeric variable.

```
PROC FORMAT;  
VALUE REGNAME  
1='WEST'  
2='SOUTH'  
3='MIDWEST'  
4='NORTH'  
5='INVALID';
```

```
PROC FREQ;  
TABLES REGION/NOCUM NOPERCENT;  
FORMAT REGION REGNAME.;;  
RUN;
```

The results are:

<u>REGION</u>	<u>FREQUENCY</u>
WEST	1
SOUTH	2
MIDWEST	1
INVALID	1

EXTRACTING VALUES FROM EXISTING VARIABLES

The rest of the presentation shows many of the functions that SAS has included for working with character data. How do we get STATE and ZIPCODE from ADD2? Fortunately, when you looked at your observations you see that all ADD2 lines were entered in the form "CITY, STATE ZIPCODE". The comma and blank between CITY and STATE and the blank between STATE and ZIPCODE are important. STATE is a 2-character postal code. Some ZIPCODEs were entered as 5 characters while others are in the ZIP+4 FORMAT. This difference will not cause a problem but other differences might. If some observations had been entered in a different form, you might still be able to do the processing but it would be more complicated. There may be times when you have no choice except to edit each observation.

The first observation is:
Raleigh, NC 23232-2222

There are actually several ways to extract STATE and ZIPCODE. The method that will be shown later for last name could also be used here. The SCAN function will break a string into "words" based on the delimiters.

```
WORD1=SCAN(ADD2,1,',');  
WORD2=SCAN(ADD2,2,',');
```

The above code will break ADD2 into words using a comma as the delimiter. The first word will be characters up to the first comma but not including the comma. The second word will be everything after the first comma up to the second comma. In this case, there is no second comma in ADD2 so it will be to the end of the string. If you asked for a third word, you would get a blank.

You can specify one or more delimiters or use the defaults. You must put the delimiter(s) in quotes or SAS will think you are using a variable name (, is not a valid variable name so you will really have an error). Do not put any variable names in quotes or SAS will think you want the string "ADD2" scanned rather than the value of the variable.

For the first observation the results are:

```
WORD1=Raleigh
WORD2= NC, 23232-2222
```

PROC CONTENTS will reveal a characteristic of the SCAN function that will often cause problems although it does not in this case. Variables created with a SCAN function will always have a length of 200 unless you specify a length before you use the function.

To separate WORD2 into STATE and ZIPCODE, we will use the SUBSTR function. It extracts a "subset" of a string based on the positions you specify.

```
STATE=SUBSTR(WORD2,1,2);
ZIPCODE=SUBSTR(WORD2,4);
```

STATE is created by extracting 2 characters from WORD2 starting at position 1 in the string. ZIPCODE is created by starting at position 4. Since the number of characters was not specified, all characters through the end of the string will be extracted. The starting position is required but the number of characters to extract is not. The results are:

```
STATE= N ZIPCODE=23232-2222
```

Why did ZIPCODE work but not STATE? Actually neither one is what you might think. The blank between the comma and the state code caused the problem. The first and fourth characters of WORD2 are both blanks and SUBSTR counted them. ZIPCODE actually starts with a blank. To solve this problem you can either change your SUBSTR function or use another function first to eliminate any lead blanks. The second method is safer because you never know when you will have 2 or more lead blanks.

```
WORD2=LEFT(WORD2);
```

This code will left-justify the string and eliminate any lead blanks. You would then follow this code with the SUBSTR function statements. The following

code will give you correct results.

```
WORD1=SCAN(ADD2,1,',');
WORD2=LEFT(SCAN(ADD2,2,','));
STATE=SUBSTR(WORD2,1,2);
ZIPCODE=SUBSTR(WORD2,4);
```

Someone might ask why we didn't use a comma and blank as the delimiters in the SCAN function. ZIPCODE would then be the third word and the blanks would not be a problem because delimiters do not appear in the words from SCAN. This would work for our first example but it would not work for

```
San Francisco, CA 99494
```

In this case, WORD1 would be "San", WORD2 would be "Francisco" and WORD3 would be "CA" with ZIPCODE being the fourth word. There are even some cities with three words in the name.

One other thing that you noticed in reviewing your data was that "San Francisco" was incorrectly typed with two blanks instead of one. Fortunately there is now a function to fix that problem.

```
CITY=COMPBL(CITY);
```

The COMPBL (compress blanks) function will compress multiple blanks into a single blank. If you wanted to remove all the blanks you would use the COMPRESS function.

```
CITY=COMPRESS(CITY);
```

Too bad we can not use the same logic for extracting last name because there is not a pattern – or is there? Did you notice that the last name occurs after the last blank? But how do you find the last blank? There is not a SCAN from the right function but there is a REVERSE function.

```
REVNAME=REVERSE(NAME);
```

The reversed names are:

```
nosleN yenraB
taogiluK ydnA
kcuB naV beD
tebbiR hpesoJ .S
HTROW NEROL
```

It is obvious there are lead blanks since the names are right-aligned but SAS prints character variables as left-aligned. NAME was created with a length of 30

TUTORIALS

so any name with less than 30 characters will have blanks added to make it 30. The trailing blanks are now lead blanks when the REVERSE function is used. PROC PRINT will actually drop off the lead blanks that are common to all the values but you will see them if you use FSVIEW.

Now that you have REVERSED the string, you can use SCAN to break REVNAME into first and last names. When there are several delimiters together, they will be treated as one. If the first character is a delimiter, it is ignored. You will need to remember to type in an actual blank in the SCAN function to specify the delimiter. The length is specified so you do not end up with 200 characters. Although the length could be shorter, I used the same as the original variable to ensure there were not any people with a single name which happened to be 30-characters (we have Cher, Madonna and John-Boy so why not David-Joseph-Andrew-GregoryBob).

```
LENGTH REVNAME LASTNAME REST_NAME
$30;
```

```
REVNAME=REVERSE(NAME);
LASTNAME1=SCAN(REVNAME,1,'');
LEFTNAME1=LEFT(REVNAME);
Z=INDEX(LEFTNAME1,'');
REST_NAME1=SUBSTR(LEFTNAME1,Z);
```

The name is reversed as shown above. SCAN is used to get the first word from the reversed name so you will end up with the last name. Getting the rest of the name is a little trickier because you need to account for people like "S. Joseph Ribbet". You can simply select the second word because that would only give you "Joseph". The LEFT function will get rid of the leading blanks. The INDEX function returns the location of the specified character. In the examples of

```
nosleN yenraB
taogiluk ydnA
```

the first blanks are the 7th character and 9th character, respectively. You then use these results as part of the parameters for your SUBSTR function. Earlier we specified a specific number for the starting point of the substring function and it applied to all observations. In this example, the starting point will be specific to each observation. You will end up with:

NAME	LASTNAME1	REST_NAME1
Andy Kuligoat	taogiluk	ydnA
Deb Van Buck	kcuB	naV beD
S. Joseph Ribbet	tebbiR	hpesoJ .S

```
LOREN WORTH      HTROW      NEROL
```

Now you just need to reverse them back. Do not forget to left-justify them, too. So you have

```
LASTNAME=LEFT(REVERSE(LASTNAME1));
FIRSTNAME=LEFT(REVERSE(REST_NAME1));
```

Although it wasn't explicitly request, you are also going to assume the names need to be in the form "last name, first name". But first, you noticed that Loren Worth was all capital letters and you really want mixed case. There are UPCASE and LOWCASE functions. You just need to separate out the first letter and UPCASE it while using LOWCASE on the rest of the letters. The following code should accomplish this and put everything back together again in the desired form.

```
LENGTH FIRSTLTR LASTLTR $1;
FIRSTLTR=
  UPCASE (SUBSTR (FIRSTNAME, 1, 1) );
FIRST_REM=
  LOWCASE (SUBSTR (FIRSTNAME, 2) );
LASTLTR=
  UPCASE (SUBSTR (LASTNAME, 1, 1) );
LAST_REM=
  LOWCASE (SUBSTR (LASTNAME, 2) );
NEWNAME=
  LASTLTR || TRIM (LAST_REM) ||
  ', ' ||
  FIRSTLTR || FIRST_REM;
```

The || is the concatenate symbol. Sometimes finding this on your keyboard is the hardest part about writing the code. You may need to check your keyboard mapping to find the right keys. Your final results are:

```
Nelson, Barney
Kuligoat, Andy
Buck, Deb van
Ribbet, S. joseph
Worth, Loren
```

You will notice that everything looks fine except for the two people that really had three part names. You have a couple options and which is most appropriate will depend upon your situation. You could write additional code similar to the above code that would check for 3 or even 4-part names. But what if you have a name like "Dr. John Jacob Wizehimer III"? The possibilities are really endless. You need to decide if having a few wrong cases is worth the effort to try to identify everything. Another option is to simply identify the observations that did not have the

standard 2-part name and edit these by hand. This can be done easily by using SCAN to see if the third word is not blank. In my case, this has been the best solution because I only had 4 or 5 special cases out of 30,000 observations. One other thing to consider is the case of Deb Van Buck. Is "Van" really her middle name or is her last name "Van Buck". How do you know which of the following is the correct result?

Buck, Deb Van
Van Buck, Deb

The next step is to process the INTEREST variable. The requester wanted to know how many people were interested in each subject area. A quick PROC FREQ would seem to give you the answer. However, people have multiple interests so you need to account for each one. Just like you have before, you can use SCAN to break the variable into individual words. The difference this time is that the new variable will have the same variable name for the different words and you will use an OUTPUT statement after you create each word. The OUTPUT statement will result in one record per word per original record.

```
NEW_INTEREST=SCAN(INTEREST,1,',');
OUTPUT;
NEW_INTEREST=SCAN(INTEREST,2,',');
OUTPUT;
... continue until you have accounted for the
maximum number of words...
```

How do you know the maximum number of words? You do not want the people with fewer interests to have blank records. The easiest way to do this is to count the number of commas. You will need to add 1 to the count since the last word will not be followed by a comma.

The LENGTH function is used to determine the actual length of a value excluding any trailing blanks. Do not confuse this with the LENGTH statement which sets the length for the variable for all observations. The COMPRESS function is used to eliminate certain characters from a string. To count the occurrences of a character, find the length of the original value and subtract the length after you have eliminated that character.

```
INTEREST_COUNT=LENGTH(INTEREST) -
LENGTH(COMPRESS(INTEREST,',')) + 1;
```

You now have an accurate count of the interests for each observation. You can now use a DO LOOP to write your statements.

```
DO I=1 TO INTEREST_COUNT;
NEW_INTEREST=
LEFT(UPCASE(SCAN(INTEREST,I,'')));
OUTPUT;
END;
```

You will notice I also put in the LEFT and UPCASE functions. LEFT will eliminate any leading blanks that might have been between the comma and the interest value while UPCASE will ensure the case is consistent across observations. However, you still have the problem where people used differently terminology for the same thing. You could use IF/THEN statements, CASE logic or PROC FORMAT to change the values. Another method is to use the TRANWRD function.

```
LENGTH NEW_INTEREST2 $30;
NEW_INTEREST2=
TRANWRD
(NEW_INTEREST,'AUDIO/VISUAL','A/V');
```

This will change the string "AUDIO/VISUAL" to the shorter string "A/V". The LENGTH statement is needed or you end up with a 200-byte field. Do not confuse this function with TRANSLATE which has been available longer. TRANSLATE changes individual letters. It also has you specify the outcome before the original value. If you attempted to use

```
TEST= TRANSLATE
(NEW_INTEREST,'AUDIO/VISUAL','A/V');
```

you would get "AUDIOUDISUAL" when the original value was "AUDIO/VISUAL". This is because all "A"s translated to "A", "I" translated to "U" and "V" translated to "D".

Just when you think you are finished, you also notice the observation which had quotes around the person's interests. You can now easily eliminate the quotes

```
INTEREST=DEQUOTE(INTEREST);
```

You can also do the opposite with the QUOTE function. The latter is helpful if you will be writing your data to a file so it can be read with other software. A primary use is when you need to write a comma-separated file but your data happens to have commas in it. By using quotes around the entire value, the embedded commas will be ignored.

The requester also wanted to know if everyone provided a legitimate ZIPCODE and how many

TUTORIALS

provided the ZIP+4 version. To determine if the ZIPCODE is a numeric or a numeric with a dash, you can use the verify function:

```
CHECK=VERIFY(ZIPCODE,'0123456789-');
```

If the only characters are the ones specified, then the variable CHECK will have a value of 0. Otherwise, it will have the position of the first character which was not specified. You must include the blank because observations with a 5-digit code rather than ZIP+4 will have a value of 6 since the 6th–10th characters are blank.

OTHER FUNCTIONS

Believe it or not, there are even more character functions. The above examples used INDEX. There is also INDEXC and INDEXW functions. INDEXC looks for any character while INDEXW looks for the entire word. For example, the data has observations with interests for

Data warehousing, a/v, food

logistic regression, data mining

If you look for "housing", INDEX will find it at position 10 in the first record and will not find it in the second. INDEXW will not find it in either record because it looks for an entire word rather than a partial word. You also need to note that a comma immediately after the desired string will prevent INDEXW from finding it. INDEXC will return a 10 for the first record a 2 for the second record. You might wonder how this is possible since 'housing' is not in the second record. While the entire word is not there, the letter 'o' is indeed in the second position.

There is also a RIGHT function which right-justifies data just like the LEFT function. In fact, we could have used it in the above code when we were reversing name to avoid getting the lead blanks.

TRIMN is similar to TRIM but it returns a null string if the expression is missing.

There are also several functions which I have not yet had the need to use. These are listed at the end of the paper following the code.

SUMMARY

There is a lot of information here but you should be able to handle almost any character data that you might encounter. I said almost because I did not cover things like non-printable characters or fields that are too wide to be printed or viewed. When you work with character data, you should remember to:

- Use PROC CONTENTS to verify lengths
- Specify lengths for new variables
- Look for patterns to use to break up strings
- Watch out for leading and trailing blanks
- Remember there can be more than one way to accomplish the same results
- Sometimes you have to be satisfied with incorrect results or edit the data by hand

The final code and output are shown at the end of the paper. In some places, the code is different than in the body of the paper because I combined steps.

TRADEMARK INFORMATION

SAS is a registered trademark of SAS Institute, Inc. in the USA and other countries. ® indicates USA registration.

CONTACT INFO

Deb Cassidy
Cardinal Distribution
7000 Cardinal Place
Dublin, OH 43017
deb.cassidy@cardinal.com
614-757-7136

PART OF PROC CONTENTS

-----Alphabetic List of Variables and Attributes-----

#	Variable	Type	Len	Pos
2	add1	Char	30	38
3	add2	Char	30	68
4	interest	Char	60	98
1	name	Char	30	8
5	num_papers	Num	8	0

FIRST 5 OBSERVATIONS

Obs	name	add1	add2
1	Barney Nelson	953 Probe Way	Raleigh, NC 23232-2222
2	Andy Kuligoat	23134 Indianpaint Road	Chattanooga, TN 37204
3	Deb Van Buck	1524 Patty Lane	Franklinton, OH 43222-6028
4	S. Joseph Ribbet	Georgia Street, PO Box 4321	Sierra, CA 95474
5	LOREN WORTH	Nucleic Way	San Francisco, CA 99494

Obs	interest	num_papers
1	"Emerging Technologies, WEB pages"	51
2	Data warehousing, a/v, food	17
3	Training	1
4	logistic regression, data mining	8
5	data presentation, Audio/Visual, logistic regression	.

FINAL OUTPUT**REGIONS AND ZIPCODE CHECK****The FREQ Procedure**

region	Frequency
WEST	1
SOUTH	2
MIDWEST	1
INVALID	1

CHECK	Frequency
0	5

TUTORIALS

LIST OF PARTICIPANTS

Obs	NAME
1	Kuligoat, Andy
2	Nelson, Barney
3	Ribbet, S. Joseph
4	Van Buck, Deb
5	Worth, Loren

Interests After Combining for Spelling Differences

The FREQ Procedure

interest2	Frequency
A/V	2
DATA MINING	1
DATA PRESENTATION	1
DATA WAREHOUSING	1
EMERGING TECHNOLOGIES	1
FOOD	1
LOGISTIC REGRESSION	2
TRAINING	1
WEB PAGES	1

FINAL CODE

```
PROC CONTENTS DATA=ORIGINAL;  
TITLE 'ORIGINAL DATA';  
RUN;
```

```
PROC PRINT DATA=ORIGINAL;  
TITLE 'ORIGINAL DATA';
```

```
DATA FINAL;  
SET ORIGINAL;  
LENGTH REVNAME $30 FIRSTLTR LASTLTR $1 NEWNAME $31 ;  
REVNAME=LEFT(REVERSE(NAME));  
LASTNAME1=SCAN(REVNAME,1,'');  
THIRD_NAME=SCAN(REVNAME,3,'');  
FIRST_SPACE=INDEX(REVNAME,'');  
REST_NAME=SUBSTR(REVNAME,FIRST_SPACE);  
  
LASTNAME=LEFT(REVERSE(LASTNAME1));  
FIRSTNAME=LEFT(REVERSE(REST_NAME));  
  
FIRSTLTR=UPCASE(SUBSTR(FIRSTNAME,1,1));  
LASTLTR=UPCASE(SUBSTR(LASTNAME,1,1));
```

```

FIRST_REM=LOWCASE(SUBSTR(FIRSTNAME,2));
LAST_REM=LOWCASE(SUBSTR(LASTNAME,2));

NEWNAME=LASTLTR || TRIM(LAST_REM) || ',' || FIRSTLTR || FIRST_REM;

CITY=COMPBL(SCAN(ADD2,1,','));
STATE_ZIP=LEFT(SCAN(ADD2,2,','));
STATE=SUBSTR(STATE_ZIP,1,2);
ZIPCODE=SUBSTR(STATE_ZIP,4);

    IF STATE IN ('CA', 'OR') THEN REGION=1;
    ELSE IF STATE IN ('TN', 'NC') THEN REGION=2;
    ELSE IF STATE IN ('OH') THEN REGION=3;
    ELSE REGION=5;
    **** INCLUDE REST OF STATES HERE. ;

CHECK=VERIFY(ZIPCODE,'0123456789- ');

RUN;

*** NOTE THAT ANY NAMES WITH 3 OR MORE PARTS WILL BE EDITED BY HAND AT THIS POINT;

PROC FORMAT;
VALUE REGNAME
1='WEST'
2='SOUTH'
3='MIDWEST'
4='NORTH'
5='INVALID';
RUN;

PROC FREQ DATA=FINAL;
TABLES REGION CHECK/NOCUM NOPERCENT;
FORMAT REGION REGNAME.;
TITLE 'REGIONS AND ZIPCODE CHECK';
RUN;

PROC SORT DATA=FINAL;
BY NEWNAME;
RUN;

PROC PRINT DATA=FINAL SPLIT='*';
VAR NEWNAME;
LABEL NEWNAME='NAME';
TITLE 'LIST OF PARTICIPANTS';
RUN;

DATA ALL_INTEREST;
SET FINAL (KEEP=INTEREST);
INTEREST=DEQUOTE(UPCASE(INTEREST));
NUM_COMMAS=LENGTH(INTEREST) - LENGTH(COMPRESS(INTEREST,',')) + 1;
DO I=1 TO NUM_COMMAS;

```

TUTORIALS

```
INTEREST2=LEFT(SCAN(INTEREST,1,''));  
OUTPUT;  
END;  
RUN;
```

```
PROC FREQ DATA=ALL_INTEREST;  
TABLES INTEREST2/NOCUM NOPERCENT;  
TITLE 'FIRST PASS FORINTERESTS';  
RUN;
```

```
*** COMBINE INTERESTS WHICH WERE SPELLED DIFFERENTLY;  
DATA ALL_INTEREST2;  
SET ALL_INTEREST;  
INTEREST2=TRANWRD(INTEREST2,'AUDIO/VISUAL','A/V');  
RUN;
```

```
PROC FREQ DATA=ALL_INTEREST2;  
TABLES INTEREST2/NOCUM NOPERCENT;  
TITLE 'INTERESTS AFTER COMBINING FOR SPELLING DIFFERENCES';  
RUN;
```

OTHER FUNCTIONS FOR CHARACTER DATA

Byte(n)	Returns one character in the ASCII or EBCDIC sequence where n is an integer representing a specific ASCII or EBCDIC character
COLLATE(start-position<,end-position>) (start-position<,,length>)	Returns an ASCII or EBCDIC collating sequence character string
RANK(x)	Returns the position of a character in the ASCII or EBCDIC collating sequence
REPEAT(argument,n)	Repeats a character expression
SOUNDEX(argument)	Encodes a string to facilitate searching
SUBSTR(argument, position<,n>)=characters-to-replace	Replaces character value contents