SESUG Proceedings (c) SESUG, Inc (http://www.sesug.org) The papers contained in the SESUG proceedings are the property of their authors, unless otherwise stated. Do not reprint without permission.
SESUG papers are distributed freely as a courtesy of the Institute for Advanced Analytics (http://analytics.ncsu.edu).

Paper CC-028

Standardization of Lists of Names and Addresses Using SAS® Character and Perl Regular Expression (PRX) Functions

Elizabeth Heath, RTI International, RTP, NC
Priya Suresh, RTI International, RTP, NC
Ruben Chiflikyan, RTI International, RTP, NC

ABSTRACT

Survey samples are often provided as lists of names and addresses that contain numerous spellings and abbreviations for the same word. Frequently, the variations must be standardized to match application specifications before samples can be processed. Data standardization can be performed in a reproducible manner with SAS® character and PERL regular expression (PRX) functions, leaving only a small number of ambiguous cases to be handled manually. We will show how a small, fictitious survey sample can be standardized using SAS®.

KEYWORDS: PRX functions, pattern matching, SAS® functions

INTRODUCTION

Our survey sample file contains the names and addresses of people to be interviewed. Each sample member receives a letter that describes the survey and requests their participation. If the letter is returned as undeliverable, we attempt to find their new address. Applications, that generate the letter and track current addresses, process standardized data and do not process original sample data, that may contain numerous spellings and abbreviations for the same word.

In this paper we show two ways to standardize data. The first way uses SAS® character functions to break down a name field into its sub-parts of title, full name, and suffix, while the second way uses PERL regular expression functions to break down an address field into its sub-parts of street address, city, state, and zip. PERL regular expression functions are very powerful in their pattern matching capabilities. In addition to breaking down the fields they can help validate the sub-parts, if necessary.

USING SAS® CHARACTER STRINGS

Below we demonstrate how to extract titles, suffixes, and full names from fictitious survey sample names, when some names do not have titles and/or suffixes. The program retrieves and standardizes titles, suffixes, and full names using character functions of the data step. Basically, the FIND function is used to locate blank spaces that separate the first and second word and last two words in the name string. Once the spaces are located, the title, full name, and suffix can be extracted and standardized.

```
data zero;  *write names to SAS® dataset zero;
infile datalines pad missover ;
input str1 $30.;
datalines;
Mr Dave Smith Jr.
Granny Jane PHD
DR Susan B. Anthony
Doctor Ringo Smith Sr.
Mary Jane Smith Jones
Mister. Fred Smith IV
Mrs Mary Jane Smith Jones
Fred G. Sandford
;
data three(keep=str1 str2 title suffix full_name locstr1 locstr2);
length word1 word2 $10 suffix title $5 full_name $30;
set zero;
*remove leading and trailing blanks. replace multiple blanks with one blank;
str1=left(trim(compbl((str1))));
*capitalize letters in string str2 for text matching in word1 and word2;
```

```
str2=upcase(str1);
*find the titles if they are present;
*use find function to get position of the first blank.
*if no blank is found, a value of 0 is returned;
locstr1=find(str2,' ');
word1=trim(left(substr(str2,1,locstr1))); * the first word in the sting;
if locstr1 > 0 then
    do;
         *assign title values;
         if word1 in ('MR','MISTER','MISTER.','MR.') then title = 'Mr.';
         else if word1 in ('MRS','MRS.') then title = 'Mrs.';
         else if word1 in ('MS.','MS') then title = 'Ms.';
         else if word1 in ('MISS.','MISS') then title = 'Miss.';
         else if word1 in ('DOCTOR','DOCTOR.','DR','DR.') then title = 'Dr.';
         else  title='N/A';
    end;

*find the suffixes if they are present;
lng=length(str2); * the length of string;
locstr2=find(str2,' ',-lng); * the position of the last blank;
word2=trim(left(substr(str2,locstr2))); * the last word in the sting;

if locstr2 > 0 then
    do;
         *assign suffix values;
         if word2 in ('JR.','JR') then suffix = 'Jr.';
         else if word2 in ('SR.','SR','SENIOR') then suffix = 'Sr.';
         else if word2 in ('PHD','P.H.D','PH.D','P.H.D.') then suffix = 'PhD.';
         else if word2 in ('PROF','PROFESSOR') then suffix = 'Prof.';
         else if word2 in ('I','II','III','IV','V') then suffix = word2;
         else  suffix='N/A';
    end;

*find fullname in all 4 various scenarios;
     if title='N/A' and suffix='N/A' then full_name=substr(str1,1,lng);
     else if title ^='N/A' and suffix='N/A' then
     full_name=substr(str1,locstr1,lng-locstr1+1);
     else if title ='N/A' and suffix ^='N/A' then
     full_name=substr(str1,1,locstr2);
     else if title ^='N/A' and suffix ^='N/A' then
     full_name=substr(str1,locstr1,locstr2-locstr1+1);
     full_name=trim(left(full_name));
run;
```

The results can be seen Figure 1.

Obs	suffix	title	full_name	str1
1	Jr.	Mr.	Dave Smith	Mr Dave Smith Jr.
2	PhD.	N/A	Granny Jane	Granny Jane PHD
3	N/A	Dr.	Susan B. Anthony	DR Susan B. Anthony
4	Sr.	Dr.	Ringo Smith	Doctor Ringo Smith Sr.
5	N/A	N/A	Mary Jane Smith Jones	Mary Jane Smith Jones
6	IV	Mr.	Fred Smith	Mister. Fred Smith IV
7	N/A	Mrs.	Mary Jane Smith Jones	Mrs Mary Jane Smith Jones
8	N/A	N/A	Fred G. Sandford	Fred G. Sandford

Figure 1. The extracted and standardized values for title, suffix, and full_name variables

USING PERL REGULAR EXPRESSION (PRX) FUNCTION CALLS

It is a two step process to use PRX function calls for string parsing. First the pattern to be recognized is defined and then the appropriate function is selected to match the pattern in the string. Once the pattern is matched, it is easy to break up the string.

Fictitious sample address information is shown below. Note that the street address and city are separated by a comma and are stored in the variable called "ADDRESS".

903 Saint Stevens Court, Raleigh NC 27221
6 Firststreet Street apt 3, High Point SC 55555
925A Raleigh Terrace #11, Detroit MI 66666
9 Leadsville Avenue Buzzer 1, Laramie WY 77777

EXTRACTING STATE AND ZIP CODE

The following example shows how pattern matching can be done in order to extract the 2-character state abbreviation and 5-digit zip code:

```
 patternStZipExpression = "/[A-Z][A-Z][\t ]+\d{5}/i";              (a1)
patternStZipId = PRXParse (patternStZipExpression);                (b1)
call PRXSubStr (patternStZipId, ADDRESS, position, length);        (c1)
if position ^= 0 then do;                                          (d1)
  matchedSubString = SubStr (ADDRESS, position, length);           (e1)
   stateAbbr = SubStr (matchedSubString, 1, 2);                    (f1)
   Zip = trim(substr (matchedSubString, 4));                       (g1)
end;                                                               (h1)
```

The PERL regular expression to match the state and ZIP code is given in line (a1). In this expression, /i makes the match case-insensitive (e.g. both NC and nc will be matched), [A-Z][A-Z] specifies that there are two consecutive alphabetic characters, [\t]+ specifies that there might be one or more spaces or tabs, and \d(5) specifies that there must be 5 digits. In line (b1), the function PRXParse loads the pattern into memory and returns an integer ID that is used by other PRX functions as a reference to the pattern. This should be done only once per data step (e.g. in an if _N_ = 1 block). Function PRXSubStr returns the position and length of the defined state and zip code pattern in string ADDRESS, in line (c1). If the pattern is matched, position will have a value greater than 0. When the position is matched, the state and zip code can be extracted using the substring function, as shown in lines (d1) through (h1).

EXTRACTING PARTS OF THE ADDRESS

Code for extracting other parts of the address string is shown in lines (i1) through (n1). Lines (i1) and (j1) show the PERL regular expression for matching a street number and street name, where the street name has multiple words (including the special character "#") and ends with a comma delimiter. In this example, the street address is parsed out first from the ADDRESS variable. Then from the string that contains the street address, the street number is teased out.

This PERL expression will match the street address which consists of the street number and street name:

```
patternStreetAddr = "/(\d+)[\s\w\#]+,/i";                          (i1)
patternStreetAddrId = prxparse(patternStreetAddr);                 (j1)
```

The following PERL expression will match just the street number in the string that contains the street address:

```
patternStreetNum = "/(\d+)\w*/i";                                  (k1)
patternStreetNumId = prxparse(patternStreetNum);                   (l1)
```

The comma delimiter is used to match the city, state, and zip code in the original address string in lines (m1) and (n1). The state and zip code can be extracted following the code that was previously described in lines (a1) through (h1), and then the city can be extracted.

```
patternCityStZip = "/,(\s+\w+)+/i";                                (m1)
patternCityStZipId = prxparse(patternCityStZip);                   (n1)
```

Below we demonstrate how to extract address components using the PRX pattern matching, and show the components versus the original string in Figure 2.

SAMPLE ADDRESS EXTRACTION CODE

```
*write addresses to a SAS® dataset;
data addresses2;
  input address $55.;
  datalines;
   903 Saint Stevens Court,  Raleigh NC 27221
   6 Firststreet Street apt 3, High Point SC 55555
   925A Raleigh Terrace #11, Detroit MI 66666
   9 Leadsville Avenue Buzzer 1, Laramie WY 77777
   ;
data addresses;
set addresses2;
length
 address $55.
StreetNum $10.
StreetName $40.
CityName $30.
StateName $2.
Zip $9.
;
*remove leading and trailing blanks.
 replace multiple consecutive blanks with one blank;
address = left(trim(compbl(address)));

*define patterns;
if _N_ = 1 then
do;

   patternStateZip = "/[A-Z][A-Z][\t ]+\d{5}/i";
   pattern_StateZip = prxparse(patternStateZip);

   patternAddress = "/(\d+)[\s\w\#]+,/i";
   pattern_Address = prxparse(patternAddress);

   patternStreetNum = "/(\d+)\w*/i";
   pattern_StreetNum = prxparse(patternStreetNum);

   patternCity = "/,(\s+\w+)+/i";
   pattern_City = prxparse(patternCity);

end;

retain pattern_StateZip pattern_Address pattern_City pattern_StreetNum;

/* pattern_StateZip is state zip */
call prxsubstr(pattern_StateZip, ADDRESS, position, length);
   if position ^= 0 then
   do;
     match = substr(ADDRESS, position, length);
     statename = substr (match, 1, 2);
     zip = substr(match, 4);
   end;

/* pattern_Address is Street number, name, and apt info */
call prxsubstr(pattern_Address, ADDRESS, position, length);
   if position ^= 0 then
   do;
      match = substr(ADDRESS, position, length);

   call prxsubstr(pattern_StreetNum, match, position2, length2);
```

```
      if position2 ^= 0 then
      do;
         match2 = substr(match, position2, length2);
         StreetNum = match2;
         streetname = trim(compress(substr(match, length2+1 ),','));
      end;
      else
       do;
                streetname = match;
       end;
   end;

call prxsubstr(pattern_City, ADDRESS, position, length);
   if position ^= 0 then
   do;
      match = substr(ADDRESS, position, length);

   call prxsubstr(pattern_StateZip, match, position2, length2);
      if position2 ^= 0 then
      do;
         match2 = substr(address, position, position2-1);
        cityname = trim(compress (match2, ','));
      end;
      else
      do;
       cityname = trim(compress (match, ','));
      end;
   end;
run;
```

Here are the results:

Obs	StreetNum	StreetName	CityName	StateName	Zip	address
1	903	Saint Stevens Court	Raleigh	NC	27221	903 Saint Stevens Court, Raleigh NC 27221
2	6	Firststreet Street apt 3	High Point	SC	55555	6 Firststreet Street apt 3, High Point SC 55555
3	925A	Raleigh Terrace #11	Detroit	MI	66666	925A Raleigh Terrace #11, Detroit MI 66666
4	9	Leadsville Avenue Buzzer 1	Laramie	WY	77777	9 Leadsville Avenue Buzzer 1, Laramie WY 77777

Figure 2. Extracted address components versus original address string.

VERIFYNG DATA EXTRACTION

One way to check that data components have been extracted correctly involves comparing the original and extracted data, where the comparison is made on uppercase strings that have been compressed to remove blank spaces and comma delimiters. Example compression code is shown in line (o1) for the original string and in line (p1) for the extracted data components. If the two strings, orig_str and extracted_str, match, the data extraction is correct. Incorrect data extractions can be examined manually and corrections can be programmed.

The following lines can be used for verifying the extracted data:

```
orig_str = upcase(compress(ADDRESS,',')); (o1)
extracted_str = upcase(compress((streetNum ||
                                 streetname ||
                                 cityname ||
                                 statename ||
                                 zip),',');  (p1)
```

STANDARDIZING EXTRACTED DATA COMPONENTS

After the data have been correctly extracted, the data can be formatted. For example, the application that tracks current addresses requires that the string 'street' be standardized as ‘ST’ and the string ‘road’ be standardized as

'RD'. The function TRANWRD can be used to replace every occurrence of a target string with a replacement string. In the following statement, every occurrence of the string "ROAD" is changed to "RD" in variable streetname.

```
streetname = TRANWRD(streetname,"ROAD,"RD");
```

There are instances where a TRANWRD replacement incorrectly changes the data. For example, the street name 'FIRSTSTREET STREET' should not be changed to 'FIRSTST ST'. Moreover, if an application requires that the street abbreviation 'ST' be formatted as 'STREET', it would be incorrect to use the function TRANWRD to make this change on the street name 'ST MARYS ST', where the first 'ST' string is an abbreviation for 'SAINT'. PERL regular expressions can be used to locate strings that need custom formatting. Consequently, data formatting must be carefully applied.

CONCLUSION

We presented two ways to standardize data. The first way, using SAS® character functions, searches for strings based on the position of blank spaces but does not match patterns within the strings, while the second way uses PERL regular expressions to find strings and match patterns within the strings. As shown in this paper, PERL regular expressions are very powerful pattern matching tools. They can be used not only to break down a complete address string to its essential sub-parts but also validate the sub-parts. For example, they can ensure that the zip code has exactly 5 digits (in the given example). PERL expressions require some trial and error to make sure that they work properly for all variations, or string patterns, in the data. In addition, requiring delimiters at some critical points, such as a comma between the street address and city, will make pattern matching a bit easier.

REFERENCES

SAS® Version 9 On-line Documentation

ACKNOWLEDGMENTS

We gratefully acknowledge support from Jean Richardson, who reviewed the document and suggested improvements; Nick Kinsey and Behnaz Whitmire for proofreading; and Deanna Penick for preparing the PDF version of this document.

CONTACT INFORMATION

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

Elizabeth Heath
RTI International
PO Box 12194
RTP NC 27709-2194
Phone: (919) 485-2786
E-mail: eah@rti.org

Priya Suresh
RTI International
PO Box 12194
RTP NC 27709-2194
Phone: (919) 541-7428
E-mail: psoh@rti.org

Ruben Chiflikyan
RTI International
PO Box 12194
RTP NC 27709-2194
Phone: (919) 541-6064
E-mail: rchiflikyan@rti.org

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.