

Paper CC-033

Creating Order from Chaos Using SAS® Functions

Kristine L. Dougherty, Florida Department of Corrections, Tallahassee, FL

Vicky Feldman, Florida Department of Corrections, Tallahassee, FL

ABSTRACT

SAS® analysts may have to cope with data sources with less than optimal design. An example: addresses stored in free text fields on a mainframe. This paper shows the program written to convert the free text field into street address, city, state, and zip code, using a variety of SAS functions and Perl regular expressions, all available in SAS® version 9.1.3.

INTRODUCTION

SAS programmers often are the ones destined to take less than ideal formatted data and create from it both meaningful reports or analyses and data to be used by customers in formats such as Excel or databases. SAS functions, including the relatively new access to Perl regular expressions, provide many tools that help to transform (more or less) chaotic data into ordered data fields that the customers desire. In this case, inmate "release plans" – the locations where they say they will reside when released from prison – are, for historical reasons, stored in a mainframe database in free text fields, rather than in set fields for city, state, country, and zip code. Yet customers, including law enforcement agencies requesting information on released or soon to be released inmates, prefer data that can be imported into their own databases in set fields. This paper documents some of the SAS functions that make this task easier, eliminating much manual cleanup before lists or reports are sent to customers.

CHAOTIC DATA EXAMPLES

Cleaning these data without purchase of data cleaning/quality software (an additional licensing expense) posed many challenges. Some examples of the comment block that is supposed to hold the city, state and zip code show the variety of styles used to enter the information:

FLORIDATATE/ZIP)
N.LITTLE ROCK, ARKAN72117
TBD
COSTA MESA, CA 92627-7101
PUERTOCORTES HONDURAS3310
MIAMI,FL33172
BIRMINGHAM,ENGLAND B219HH
MIAMI, FLA. 33170
PAROLE MIAMI, FL
MIAMI,FL 33145P)
LAUDERDALE LAKES, FL33319
MADISON,MICHINGAN 48071
TALLAHASSEE,FL32312 APT 1234C
MIAMIFL33170

These examples show some of the problems encountered, which include:

- All or part of the sample text for the comment block that remains after users enter data remains: (CITY/STATE/ZIP).
- Zip codes are entered as either 5 digit or 9 digit, with or without the dash for the 9 digit.
- Some zip codes are entered with 4 or 6 digits instead of 5.
- Some records have information at the end that looks like a zip code, but isn't.
- The last information may not be a zip code (may be an apartment number or country code or just a state)
- States are entered as entire state names, common abbreviations (FLA), or two-digit state postal codes
- Information is entered with a variety of delimiters, or none (commas, dashes, slashes, parentheses)

- Incomplete information is entered (only city and state)
- Misspellings are common (MICHINGAN rather than MICHIGAN)
- Extra characters are common (especially parentheses at the beginning or end, surrounding the former sample text)
- City and state may be run together without spaces or other delimiters

SOME HELPFUL SAS FUNCTIONS: TRANWRD AND TRANSLATE

Here are some of the functions, and some Perl regular expressions, that are used to clean up this data and make it presentable, and more ordered, for customers.

TRANWRD

The TRANWRD function replaces or removes all occurrences of a give word or pattern of characters that is separated by spaces or other delimiters within a character string: The syntax is TRANWRD(field,'from','to');.

TRANWRD comes in handy for getting rid of the sample text starting with the entire sample text and working backwards, although several statements may be necessary to get all the variations. For example:

```
PLNRES3_REC=TRANWRD(PLNRES3_REC, '(CITY/STATE/ZIP)', ' ');
PLNRES3_REC=TRANWRD(PLNRES3_REC, 'CITY/STATE/ZIP', ' ');
PLNRES3_REC=TRANWRD(PLNRES3_REC, 'TY/STATE/ZIP', ' ');
PLNRES3_REC=TRANWRD(PLNRES3_REC, '/STATE/ZIP', ' ');
PLNRES3_REC=TRANWRD(PLNRES3_REC, 'TATE/ZIP', ' ');
PLNRES3_REC=TRANWRD(PLNRES3_REC, 'E/ZIP', ' ');
PLNRES3_REC=TRANWRD(PLNRES3_REC, 'P)', ' ');
```

For example , the code above changes FLORIDATATE/ZIP) to FLORIDA

And MIAMI, FL.E/ZIP) becomes MIAMI, FL. However, ONLY those parts of this sample text that cannot reasonably be part of actual addresses should be replaced. For example, CITY might be part of Iowa City, a legitimate address component.

TRANWRD is also used to translate the complete state name, or common misspellings, into the standardized 2 digit postal codes for states:

```
PLNRES3_REC=TRANWRD(PLNRES3_REC, 'CALIFORNIA', 'CA');
PLNRES3_REC=TRANWRD(PLNRES3_REC, 'CALIF', 'CA');
PLNRES3_REC=TRANWRD(PLNRES3_REC, 'CALIF.', 'CA');
```

SAN DIEGO, CALIF becomes SAN DIEGO, CA

Finally, it is used to translate 2 digit state codes with periods into the 2 digit codes:

```
PLNRES3_REC=TRANWRD(PLNRES3_REC, 'N.J.', 'NJ');
```

This modifies CAMDEN, N.J. 08104 to CAMDEN NJ 08104.

TRANSLATE

After these clean-ups, some of which are more easily recognized with the special characters included, the TRANSLATE function is used to eliminate other "junk" and replace it with spaces. Unlike TRANWRD, the syntax is (field,' to', 'from').

```
plnres3_rec=TRANSLATE(plnres3_rec," ","");
plnres3_rec=TRANSLATE(plnres3_rec,';',' ');
plnres3_rec=TRANSLATE(plnres3_rec,'(',')');
plnres3_rec=TRANSLATE(plnres3_rec,' ',' ');
plnres3_rec=TRANSLATE(plnres3_rec,' ','{');
```

```

plnres3_rec=TRANSLATE(plnres3_rec,' ');
plnres3_rec=TRANSLATE(plnres3_rec,'[f]');
plnres3_rec=TRANSLATE(plnres3_rec,'[j]');
plnres3_rec=TRANSLATE(plnres3_rec,'[/]');
plnres3_rec=TRANSLATE(plnres3_rec,'');
plnres3_rec=TRANSLATE(plnres3_rec,'');

```

This removes the various parentheses and brackets, semi-colons, etc. from the text, and replaces them with spaces, again to standardize and assist in further division into zip code and state.

PERL REGULAR EXPRESSIONS: FINDING THE ZIP CODE IF IT EXISTS

This code looks for a set of numbers that matches a pattern. It will do so up to 5 times, since sometimes there are multiple patterns that look like a zip code. The ZIP59_PATTERN definition uses Perl regular expressions that indicate that the pattern must be either a 9 digit number with a dash before the last 4 digits or 5 digits not following another number but followed by a white space character. The same pattern applies when taking into account the possibility of a 4 or 6 digit number mistakenly entered for a 5 digit zip.

The basic strategy is to define the pattern with the PRXPARSE statement and retain that pattern, then call the pattern with the PRXNEXT statement.

PRXPARSE: Compiles a Perl regular expression (PRX) that can be used for pattern matching of a character value.

PRXNEXT: Returns the position and length of a substring that matches a pattern and iterates over multiple matches within one string.

If more than one pattern is found that looks like a zip code, the last one is used.

The Perl expressions also allow you to zero in on the beginning and ending of the zip code. We can then make certain assumptions about the remainder of the fields we hope to create:

If the zip code appears to start in the 1st column, then there is no city or state. Otherwise, the city and state begin at 1 and extend until the zip code starts, and then pick up again after the zip code until the end of the field (length = 30), just in case someone added something like an apartment number after the zip code.

```

data testnew3;
length zip1-zip5 $10. citystate $25.;
array zip(5) $ zip1-zip5;
array startpos(5) startpos1-startpos5;
array len(5) len1-len5;
*ESTABLISH THE POSSIBLE PATTERNS FOR THE ZIP CODE;
*must not have a non white space character after it;
*if 5,6 or 4 digit, must not follow another number;
RETAIN ZIP59_PATTERN;
set testnew;
IF _N_=1 THEN
do;
ZIP59_PATTERN=PRXPARSE("/\d\d\d\d\d\d\d\d\d\d\s{(<!\d)\d{5}\s{(<!\d)\d{4}\s{(<!\d)\d{6}\s/");
end;
start=1;
stop=length(plnres3_rec)+1;
call prxnext(zip59_pattern, start, stop, plnres3_rec, position, length);
do i=1 to 5 while (position > 0);
zip{i} = substr(plnres3_rec, position, length);
startpos(i)=position;
len(i)=length;
call prxnext(zip59_pattern, start, stop, plnres3_rec, position, length);

```

```

end;

if zip2 ne '' then
do;
zip_code=zip2;
startzip=startpos2;
lenzip=len2;
end;
else
do;
zip_code=zip1;
startzip=startpos1;
lenzip=len1;
end;
if zip_code ne '' and startzip eq 1 then
do;
CITYSTATE=' ';
end;
else if zip_code ne '' and startzip gt 1 then
do;
CITYSTATE=substr(plnres3_rec,1,startzip-1)||" "||substr(plnres3_rec,startzip+lenzip-1,30-
(startzip+lenzip)+1);
end;
ELSE CITYSTATE=PLNRES3_REC;
run;

```

An example: 55212 LAS VEGAS NV 89155. In this case, the 55212 could conceivably be a zip code, but inspection of this one (and others) shows that the last is normally the zip code, while in this case the 55212 is the continuation of a P.O. Box address from the prior street address field. The code above correctly identifies the two possible strings. STARTPOS1 = 1, LEN1=6, STARTPOS2=20, LEN2=6, and so the zipcode becomes 89155, and the remainder of the string becomes the city and state.

MORE FUNCTIONS TO FIND THE STATE IF IT EXISTS – REVERSE, LENGTH, STNAME, SUBSTR

We've now got a string that should include the city and state. The easiest thing to do now is try to find the 2 digit state code, which ideally will be at the end of this string, preceded by a space. While we could use Perl regular expressions again, here we use the REVERSE and LENGTH functions. First we create the reverse of the entire city and state field:

```
revcitystate=left(trim(reverse(citystate)));
```

For use later on we get the length of the entire city and state field:

```
length_citystate=length(citystate);
```

We also create a field that we can use to check to see if we get a valid state name using the STNAME function on the last 2 digits of the city and state field:

```
statename=stname(substr(citystate,length_citystate-1,2)) ;
```

Then we begin to break down the citystate field into city and state.

```

if statename ne '' and substr(revcitystate,3,1)=' ' then
do;
city=substr(citystate,1,length_citystate-3);
state=substr(citystate,length_citystate-1,2);

```

```
end;
```

Unfortunately, even if the last 2 characters of the field look like a state, they might actually be part of the city name, with no state name. For example, MI might well be the last part of MIAMI. So except for a few cases, we don't want to assume that it is a state code, unless the 2 characters follow a space or other delimiter that we have recoded to a space.

A few examples:

MANCHESTER ENGLAND

Reading the last 2 characters of the citystate field would lead us to label the state (ND) as North Dakota. But since it is not preceded by a space, we do not want to take that as the state.

GREENWOOD DE

DE is a valid state code, for Delaware, and it is preceded by a space, so we accept DE as the state code, and GREENWOOD as the city.

There are a small number of states (FL is the best example) where it is unlikely that this is the ending of another word, and where it is important to get that state, rather than have it attached to the city, when the user has not entered a space or delimiter between them:

```
else if substr(revcitystate,1,2)='LF' and substr(revcitystate,3,1) ne ' ' then
  do;
    city=substr(citystate,1,length_citystate-2);
    state='FL';
  end;
```

So, for MIAMIFL city=MIAMI and state=FL.

Finally, if there isn't anything that looks like a state code, the state becomes blank and the entire citystate field is used for the city.

```
else
  do;
    city=citystate;
    state=' ';
  end;
run;
```

CHECKING THE ZIP CODE AGAINST THE STATE

Using the ZIPSTATE function, a final step is added to see if there is a match between the state indicated by the zip code and the state field.

```
zip5n=substr(zip_code,1,5)+0;
if zip5n NE . then zipstate = zipstate(zip5n);
if zipstate NE ' ' and state NE ' ' and zipstate NE state then statenomatch=1;
```

For example, W. PALM BEACH, FL 48661:

Zipstate = MI (Michigan), which does not match the state FL. So this case can be flagged as suspect, and if necessary some manual checking can be done.

CONCLUSION

SAS® functions can be used creatively to clean data. The newer Perl regular expressions, available in SAS 9, provide many new capabilities, especially for pattern matching. Details on the Perl regular expressions can be found in the "Perl Regular Expression Tip Sheet" on the SAS web site (see below). In addition, many Global Forum and

regional SAS User Group papers are helpful for learning ways to utilize regular expressions.

RECOMMENDED READING

Bahler, Caroline. 2001. "Data Cleaning and Base SAS Functions." *Proceedings of the 26th Annual SAS® User Group International Conference*. May 2001.

Borowiak, Kenneth W. 2007. "Perl Regular Expressions 102." *Proceedings of the SAS® Global Forum 2007*.

Cassell, David L. 2007. "The Basics of the PRX Functions." *Proceedings of the SAS® Global Forum 2007*.

Cody, Ron and Robert Wood Johnson. 2004. "An Introduction to Perl Regular Expressions in SAS 9." *Proceedings of the 29th Annual SAS® User Group International Conference*. May 2004.

Hoyle, Larry. 2004. "Perl regular expressions." Kansas City Area SAS User Group. October 5, 2004.
<http://www2.ku.edu/~ipssr/ksdata/sashttp/kcasug2004-10/kcasug2004-10-05Hoyle.ppt>

Hu, Wenyu and Zhhang, Liping. 2007. "Using PRX to Search and Replace Patterns in Text Strings." *Proceedings of the 2007 Southeast SAS User Group Conference*.

McGowan, Kevin. 2006. "Perl Regular Expressions in SAS." *Proceedings of the 2006 Southeast SAS User Group Conference*.

Pless, Richard. 2004. "An Introduction to Regular Expressions with Examples from Clinical Data." *Proceedings of the 29th Annual SAS® User Group International Conference*. May 2004.

SAS9® – Perl Regular Expressions Tip Sheet. http://support.sas.com/rnd/base/datastep/perl_regexp/regexp-tip-sheet.pdf

CONTACT INFORMATION

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

Kristine Dougherty
Florida Department of Corrections
2601 Blair Stone Rd
Tallahassee, FL 32399-2500
Work Phone: 850-488-1801
Fax: 850-488-1967
dougherty.kristine@mail.dc.state.fl.us

Vicky Feldman
Florida Department of Corrections
2601 Blair Stone Rd
Tallahassee, FL 32399-2500
Work Phone: 850-488-1801
Fax: 850-488-1967
feldman.vicky@mail.dc.state.fl.us

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.