

Processing Free Text With SAS Version 8

Jeremy Gurney

Jeremy.Gurney@protherics.com

Protherics PLC

Introduction

- Background
- Functions, old and new
- Regular expressions

Protherics PLC

- Focussed Immunotherapeutic business.
- Two FDA approvals in the last 12 months.
- Angiotensin vaccine for treatment of hypertension currently in clinical trials.

Version 6 to Version 8

- Version 6 had a 200 character limit for SAS data set variables.
- Version 8 has a much larger limit, 32,767 characters.
- This means that SAS can now process large chunks of free text much more easily than before.

SAS Character Functions

BYTE	LENGTH	SUBSTR
COLLATE	LOWCASE	TRANSLATE
COMPBL	QUOTE	TRANWRD
COMPRESS	RANK	TRIM
DEQUOTE	REPEAT	TRIMN
INDEX	REVERSE	UPCASE
INDEXC	RIGHT	VERIFY
INDEXW	SCAN	
LEFT	SOUNDEX	

Updated Functions

- Scan: now takes negative arguments.
 - `var=scan(text_string,-2);`
- Returns the second word in from the end of the string.

New Functions

- HTMLENCODE and HTMLDECODE
- URLENCODE and URLDECODE
- These four functions relate to the encoding schemes used for URLs and HTML.
- They allow easy interaction with text that has come from or is destined for the web.

HTMLENCODE and HTMLDECODE

SYMBOL

HTML

&

&

<

<

>

>

“

"

(space)

URLENCODE and URLDECODE

SYMBOL

ENCODING

&

%26

?

%3F

%

%25

“

%22

(space)

%20

What is a regular expression?

“an algebraic expression that denotes a regular language.”

What is a regular expression?

“The RX functions and call routines provide character-string matching functionality. That is, they enable you to search for (and, optionally, to replace) patterns or characters in a string.”

Regular Expression

- SAS regular expressions are accessed from a family of functions and call routines.
- RXPARSE
- RXMATCH, call RXCHANGE, call RXSUBSTR
- call RXFREE
- (RXPATTERN, call RXSTYLE - Experimental in v8.2)

How do you use them?

- Firstly compile the RX using RXPARSE.
 - `rx=RXPARSE(pattern);`
- Pass the RX to one of the functions.
 - `position=rxmatch(rx,string);`

Patterns

- String literals: 'Hello World'
- Alphabetic characters: \$a
- Digits: \$d
- White space: \$w
- Any character: ?
- User defined: \$('a-zA-Z0-9')

More on patterns

- $+$:one or more
 - e.g. $\$d^+$ is one or more numbers.
- $*$:zero or more
 - e.g. $\$a^*$ is zero or more letters.
- $\wedge$ or $\sim$: not
 - e.g. $\sim\text{'abcdef'}$ or $\sim\text{'a-f'}$ will allow anything other than the letters a to f.

RXPARSE

- Firstly compile the RX using RXPARSE.
 - `rx=RXPARSE(pattern);`
- This compiles the expression for later use.
- The numeric variable rx is set as a pointer to the location of the compiled expression.

RXMATCH

- Returns the position of the first character of the match or 0.
 - `position=rxmatch(rx,string);`

A Simple Example

```
data mydata;  
  infile cards trunccover;  
  length indata $201;  
  rx=rxparse("dog");  
  input indata $char201.;  
  match=rxmatch(rx,indata);  
  put match=;  
cards;  
My dog is called spot.  
run;
```

The Results

```
rx=rxparse( "dog" );  
match=rxmatch(rx,indata);  
  
"My dog is called spot."  
match=4
```

Which for this example would be more efficiently programmed as:-

```
match=index(indata, "dog" );
```

A Simple Example - Revisited

```
data mydata;  
    infile cards trunccover;  
    length indata $201;  
    rx=rxparse("dog");  
    input indata $char201.;  
    match=rxmatch(rx,indata);  
    put match=;  
cards;  
My dog is called spot.  
run;
```

A Simple Example - Revisited

```
data mydata;  
  infile cards trunccover;  
  length indata $201;  
  retain rx;  
  drop rx;  
  if _n_ =1 then rx=rxparse("dog");  
  input indata $char201.;  
  match=rxmatch(rx,indata);  
  put match=;  
cards;  
My dog is called spot.  
Shep is also a dog.  
run;
```

RXSUBSTR

- Returns the position and length and score of the matched text.
 - call rxsubstr(rx,string,position);
 - call rxsubstr(rx,string,position,length);
 - call rxsubstr(rx,string,position,length,score);

RXCHANGE

- Changes one or more substrings which match a pattern.
 - Call `rxchange(rx,times,string);`
 - Call `rxchange(rx,times,old-string,new-string);`
- `rx` must contain a 'to' clause.
 - `rx=rxparse("dog to cat");`

RXCHANGE continued

- `rx=rxparse("M1=<$a> M2=<$d> to =M2 =M1");`
- `Call rxchange(rx,1,mytext);`
 - “A1 B1” becomes “1A B1”
- `Call rxchange(rx,2,mytext);`
 - “A1 B1” becomes “1A 1B”
 - “A12” becomes “1A2” not “12A”

RXFREE

- Frees memory allocated to the parsed regular expression.
 - Call `rxfree(rx)`

A Simple Example - Revisited

```
data mydata;  
  infile cards trunccover;  
  length indata $201;  
  retain rx;  
  drop rx;  
  if _n_ =1 then rx=rxparse("dog");  
  input indata $char201.;  
  match=rxmatch(rx,indata);  
  put match=;  
cards;  
My dog is called spot.  
Shep is also a dog.  
run;
```

A bad example

```
data _null_;  
  do i=1 to 1000000;  
    rx=rxparse("foo");  
  end;  
run;
```

Uses vast amounts of memory as compiled expressions are held in memory until the completion of the data step.

A bad example

```
data _null_;  
  do i=1 to 1000000;  
    rx=rxparse("foo");  
    call rxfree(rx);  
  end;  
run;
```

Memory use is negligible as compiled expressions are deleted from memory after each loop.

Useful Examples

- `rx=rxparse("my $w \b$*/ $w is");`
 - Finds ‘dog’ in ‘My dog is called spot’.
- `rx=rxparse('"Last Trade': '</b>');`
 - Retrieves stock quote between “Last Trade” and end of bold text.
- `rx=rxparse("$<5> to ‘’");`
 - Strips mark-up from HTML.

In Conclusion

- Version 8 is a big leap forward.
- There is a lot you can do in version 8 to extract data from free text.
- I have only covered a small subset of the functionality of regular expressions.
- Experiment.

References

- SAS papers
 - Warren Sarle
 - <ftp://ftp.sas.com/pub/neural/rx612.txt>
 - <ftp://ftp.sas.com/pub/neural/patt612.txt>
- “Some Practical Ways to Use the New SAS Pattern-Matching Functions”
 - Mike Rhoads
 - <http://www2.sas.com/proceedings/sugi22/CODERS/PAPER72.PDF>