

Exploring Underutilized Capabilities in SAS® – Approaches Seldom Used

Diganta Bose,
Associate Director, FSP, Cytel

Monday, 15 Dec 2025

PRX Basics

➤ **PRX or Perl Regular Expressions – What is it exactly?**

➤ **What constitutes PRX ?**

1) **PRX Delimiters:**– examples - / / , # #, ! !, ~ ~, () , { }

2) **PRX Symbols:**– examples ^, ., +, *, |, {n}, \$, [^]

3) **Common PRX Character Classes:**- \d, \D, \s, \S, \w, \W, \b

4) **Common SAS® PRX Functions:**- PRXPARSE, PRXMATCH, PRXSUBSTR, PRXCHANGE, PRXPOSN

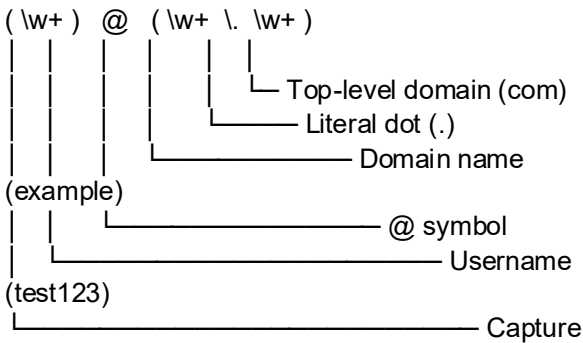
Example of a Simple Application

```
data a;
  text = "test123@example.com is my email id";
  re = prxparse('/(\w+)\@(\w+\.\w+)/');

  if prxmatch(re, text) then do;
    call prxsubstr(re, text, s, l);
    email = substr(text, s, l);
    user = prxposn(re, 1, text);
    dom = prxposn(re, 2, text);
  end;

  clean = prxchange('s/\d+/XXX/', -1, text);

  call prxfree(re);
run;
```



text	s	l	email	user	dom	clean
test123@example.com is my emailid	1	19	test123@example.com	test123	example.com	testXXX@example.com is my emailid

SAS Function	What It Extracts	Result
PRXMATCH	Confirms pattern exists	Match found
PRXSUBSTR	Full matched email	test123@example.com
PRXPOSN(re,1,...)	Capture Group 1	test123
PRXPOSN(re,2,...)	Capture Group 2	example.com

Usage in PROC SQL

What you Cannot Do ?

```
proc sql;  
select *  
  from emails where email regex  
'/\w+@\w+\. \w+/';  
quit;
```

What you can do instead ?

```
data emails2;  
  set emails;  
  re = prxparse('/\w+@\w+\. \w+/');  
  is_valid = prxmatch(re, email);  
run;
```

```
proc sql;  
  select *  
    from emails2 where is_valid > 0;  
quit;
```

Applying PRX for Data Cleaning and Harmonization

```
elementk = prxchange("s/Cyt001/DRUGX/",-1,element);  
element_B = prxchange("s/Comparative/DRUGY/",-1,elementc);
```

```
dosnum = prxchange("s/[^0-9]//",-1,element_b);  
dname = prxchange("s/[0-9]|MG//",-1,element_b);
```

ELEMENT	ELEMENT_B
Comparative 20 MG	DRUGY 20 MG
Cyt001 0.3 MG	DRUGX 0.3 MG
Cyt001 1 MG	DRUGX 1 MG
Cyt001 10 MG	DRUGX 10 MG
Cyt001 3 MG	DRUGX 3 MG

ELEMENT_B	DOSNUM	DNAME
DRUGX 1 MG	1	DRUGX
DRUGX 3 MG	3	DRUGX

Pattern matching task to cross check between SDTM SE/ TE and SDTM EX and DM in order to extract specific information to derive key treatment variables in ADSL like TRT01A, TR01SDT and TRT01EDT, e.t.c.. ?

PRX in Clinical Data Standardization

Same test represented in 5 different ways –
ALT (SGPT), S.G.P.T, Alanine Transaminase, alt level, SGPT – serum

```
/* Convert to uppercase; alternatively can use the /i operator in regEX*/
lbtest_raw = upcase(lbtest);

/* Remove special characters */
lbtest_clean = prxchange('s/[^\sA-Z ]+/ /', -1, lbtest_raw);

/* Normalize multiple spaces */
lbtest_clean = prxchange('s/\s+/ /', -1, lbtest_clean);

/* Identify ALT / SGPT using regex */
if prxmatch('/\b(ALT|S\s*G\s*P\s*T|ALANINE\s+TRANSAMINASE)\b/', lbtest_clean) then do;
    lbtestcd = 'ALT';
    lbtest = 'Alanine Aminotransferase';
end;
```



lbtest_raw	lbtest_clean	lbtestcd	lbtest
ALT (SGPT)	ALT SGPT	ALT	Alanine Aminotransferase
S.G.P.T	S G P T	ALT	Alanine Aminotransferase
ALANINE TRANSAMINASE	ALANINE TRANSAMINASE	ALT	Alanine Aminotransferase
ALT LEVEL	ALT LEVEL	ALT	Alanine Aminotransferase
SGPT - SERUM	SGPT SERUM	ALT	Alanine Aminotransferase

PRX in SAS® Macros

Identifying Group of related Variables

```
proc contents data=ads.adsl out=_m_contents_adsl(keep=memname name length type varnum) noprint;  
run;
```

- *`prxmatch(prxparse("/[\D+]fl/i"),name)>0`*
- *`prxmatch(prxparse("/trt([\d][\d])p/i"),name)>0`*
- *`prxmatch(prxparse("/tr([\d][\d])[s|e]dt/i"),name)>0`*

PRX in SAS® Macros

a macro - %m_groupage

```
%m_groupage(inds=testdata,/*Input Dataset*/  
            agegrlst=%str(<30#30-45#46-65#>65), /*age group specification*/  
            varname=agegr1,/*Age Group Variable Name*/  
            outdata=_1_sample); /*Output Dataset*/
```

*{(Below x1) or (Under x1) or (< x1) or x0 - (x1 - 1)} #
x1 - x2 #
(x2 + 1) - x3 #
(x3 + 1) - x4 #
.....
.....
(xi + 1) - xj #
(xj + 1) - xn #
{(Over xn) or (Above xn) or (>xn) or (xn + 1) - xm}*

Incorrect Entry

```
agegrlst=%str(>30#30-45#46-65#>65)
```

Acceptable Entries

1. agegrlst=%str(<30#30-45#46-65#>65) or
2. agegrlst=%str(below 30#30-45#46-65#Above 65) or
3. agegrlst=%str(Under 30#30-45#46-65#Over 65)
4. or any combination of the above 3

PRX in SAS® Macros

macro %m_groupage - correct entry example

```
agegrlst=%str(<30#30-45#46-65#>65);
```

Sample Input	
 USUBJID	 AGE
CYT001201-101-1089	56
CYT001201-102-1105	42
CYT001201-103-1131	75
CYT001201-203-1186	26

Expected Output			
 USUBJID	 AGE	 agegr1	 agegr1N
CYT001201-101-1089	56	46-65	3
CYT001201-102-1105	42	30-45	2
CYT001201-103-1131	75	>65	4
CYT001201-203-1186	26	<30	1

PRX in SAS® Macros

macro %m_groupage - checking incorrect entry

Examples of Incorrect Entries:

agegrlst=%str(>30#30-45#46-65#>65) || agegrlst=%str(< 30#31-45#46-65#>65)

First Step is to arrange age groups in order of age group ranges based on increasing Age

```
array range{&nagegrp} $100;  
array rsum{&nagegrp};  
array rord{&nagegrp} $200;  
do q = 1 to &nagegrp;  
    range{q}=strip(scan(agegrlst,q,"#"));  
    if prxmatch(prxparse("/(above|over|>)( *)\d+/i"),range{q})>0 then rsum{q} = 9999;  
    else if prxmatch(prxparse("/(below|under|<)( *)\d+/i"),range{q})>0 then rsum{q} = 1;  
    else if prxmatch(prxparse("/-/i"),range{q})>0 then  
        rsum{q} = input(strip(scan(range{q},1,"-")),best.) + input(strip(scan(range{q},2,"-")),best.);  
    rord{q} = strip(put(rsum{q},best.))||"||"||strip(range{q});  
end;
```

PRX in SAS® Macros

macro %m_groupage - checking incorrect entry example 1

agegrlst=%str(>30#30-45#46-65#>65)

2nd Step is write check programs

```
*-----*
|Below steps check incorrect entries|
*-----;
if prxmatch(prxparse("/to/i"),group_name)>0 then flag="Y";
if prxmatch(prxparse("/=/i"),group_name)>0 then flag="Y";
if &lobs > 1 then do;
  if iord = 1 and prxmatch(prxparse("/(>)|(above)|(over)/i"),group_name)>0 then flag="Y";
  if iord = &lobs and prxmatch(prxparse("/(<)|(below)|(under)/i"),group_name)>0 then flag="Y";
end;
if 1 < iord < &lobs and
prxmatch(prxparse("/(>)|(<)|(above)|(below)|(under)|(over)|(>=)|(<=)/i"),group_name)>0 then flag="Y";
if 1 < iord < &lobs and
prxmatch(prxparse("/(>)|(<)|(above)|(below)|(under)|(over)|(>=)|(<=)/i"),group_name)>0 then flag="Y";
```

ord	group_name	iord	flag
75	30-45	1	
111	46-65	2	
9999	>30	3	Y
9999	>65	4	

ERROR:----- Incorrect Entry in parameter AGEGROUP -----
----Please enter acceptable values as indicated in the user guide----



PRX in SAS® Macros

macro %m_groupage - checking incorrect entry example 2

`agegrlst=%str(<30#31-45#46-65#>65)`

2nd Step is write check programs

⊕ ord	⚠ group_name	⊕ l1	⊕ u1	⊕ ux	⊕ test_val	⚠ pass	⚠ passm
1	<30	31	29	.	0	Y	Y
76	31-45	31	45	29	29	Y	N
111	46-65	46	65	45	45	Y	Y
9999	>65	66	64	65	65	Y	Y

ERROR: [m_groupage] ----- Ranges specified in parameter AGEGRNST misses out certain values of AGE

```
if prxmatch(prxparse("/-/i"),group_name)>0 then do;
```

```
  l1 = input(strip(scan(group_name,1,"-")),best.);
```

```
  u1 = input(strip(scan(group_name,2,"-")),best.);
```

```
end;
```

```
else if prxmatch(prxparse("/(>)|(<)|(above)|(below)|(under)|(over)/i"),group_name)>0 then do;
```

```
  l1 = input(substr(group_name,anydigit(group_name)),best.) + 1;
```

```
  u1 = input(substr(group_name,anydigit(group_name)),best.) - 1;
```

```
end;
```

```
  ux=lag(u1);
```

```
if not missing(ux) then test_val = ux;
```

```
else if missing(ux) then test_val = 0;
```

```
if l1 > test_val then pass = "Y";/*Checking Overlapping Age values in AGEgroups defined*/
```

```
else pass = "N";
```

```
if l1 = 0 and ord = &ford then pass = "Y";
```

```
if ord > &ford and (l1 - ux) > 1 then passm = "N";/*Checking Skipped Age values in AGEgroups defined*/
```

```
else passm = "Y";
```

Closing Remarks

- PRX + Macros = Power & Flexibility:** Enables automated, parameter-driven handling of unstructured and semi-structured data.
- Enhanced Efficiency:** Boosts scalability, consistency, and reduces manual coding in clinical and operational workflows.
- Challenges:** Regex patterns can be complex to read, debug, and may impact performance if poorly designed.
- Best Practices:** Use modular macro design, detailed documentation, and rigorous testing to ensure stability.
- Industry Evolution:** Growing adoption of **R**, which also supports powerful regex capabilities via **PCRE functions** (e.g., grep, gsub, regexpr).

Cytel

Thank You!