The Wildcard Side of SAS®

A 10 Minute SAS® 9 Tech Tip

presented by:

Winfried Jakob, Senior Analyst, Data Quality

Canadian Institute for Health Information

Canadian Institute for Health Information
Institut canadien d'information sur la santé

Alternate Title:

Pattern Matching in SAS® 9 using Perl Regular Expressions

Agenda

- Definition of “Pattern Matching” and “Regular Expressions”
- The Mechanics of Pattern Matching in SAS® 9 by Example
- Regular Expression Basics
- “The Regex Coach” - A Learning Tool
- A More Advanced Example
- How to Get Started
- Final Words of Wisdom...

Definition 1: Pattern Matching

"Pattern matching enables you to search for and extract multiple matching patterns from a character string in one step, as well as to make several substitutions in a string in one step."[1]

Definition 2a: Regular Expressions

"Regular expressions are a pattern language which provides fast tools for parsing large amounts of text. Regular expressions are composed of characters and special characters that are called metacharacters."[1]

Definition 2b: Regular Expressions

"Perl's 'regular expressions' give you a simple language to search for patterns, extract patterns from strings, and even change text that matches your patterns. This is invaluable for all manner of text manipulation, including validation, text replacement, and string testing. With the advent of SAS® 9, the power of Perl's regular expressions is now available in the DATA step."[2]

The Mechanics of Pattern Matching

Example 1.1 String Validation

Logical Steps:

1. Define a search pattern/regular expression
2. Compile the regular expression
3. Use the regular expression to find matches

Example 1.1 String Validation

```
Listing of Example 1 and 2 Data

Obs    Text

  1    Ms. Winifred Jacob
  2    Winfred Jakob
  3    Wyn Fried Jakob
  4    Winfired Jakob
  5    Winfried Jakob
  6          Winfried Jakob
  7    Winfried  Jakob
  8    Winfried       Jakob
  9      Winfried   Jakob
 10        WinfriedJakob
```

Example 1.1 String Validation

```
data _null_;
   set Example1; * Read Example data *;
   Pattern = "/winfried jakob/i";      * Define PRX *;
   PRX = prxparse(Pattern);            * Compile PRX *;
   Match_Result = prxmatch(PRX,Text); * Search for PRX *;
   if Match_Result eq 0 then Message="No Match: ";
   if Match_Result gt 0 then Message="MATCH in: ";
   if _N_ eq 1 then do;
      putlog "The PRX pattern is: " Pattern /;
   end;
   putlog "Obs " _n_ 2.0 " " Message " " Text $char20. /;
run;
```

Example 1.1 String Validation

```
The PRX pattern is: /winfried jakob/i
```

```
Obs  1 No Match:  Ms. Winifred Jacob
Obs  2 No Match:  Winfred Jakob
Obs  3 No Match:  Wyn Fried Jakob
Obs  4 No Match:  Winfired Jakob
Obs  5 MATCH in:  Winfried Jakob
Obs  6 MATCH in:        Winfried Jakob
Obs  7 No Match:  Winfried  Jakob
Obs  8 No Match:  Winfried       Jakob
Obs  9 No Match:    Winfried   Jakob
Obs 10 No Match:      WinfriedJakob
```

Example 1.2   String Validation

```
data _null_;

   ... same code as before ...

   Pattern = "/winfried\s*jakob/i";   * Define PRX *;

   PRX = prxparse(Pattern);           * Compile PRX *;

   Match_Result = prxmatch(PRX,Text); * Search for PRX *;

   ... same code as before ...

run;
```

Example 1.2 String Validation

```
The PRX pattern is: /winfried\s*jakob/i
```

```
Obs  1 No Match:  Ms. Winifred Jacob
Obs  2 No Match:  Winfred Jakob
Obs  3 No Match:  Wyn Fried Jakob
Obs  4 No Match:  Winfired Jakob
Obs  5 MATCH in:  Winfried Jakob
Obs  6 MATCH in:        Winfried Jakob
Obs  7 MATCH in:  Winfried  Jakob
Obs  8 MATCH in:  Winfried       Jakob
Obs  9 MATCH in:    Winfried   Jakob
Obs 10 MATCH in:      WinfriedJakob
```

Example 2.1 String Replacement

```
data _null_;

   ... same code as before ...

   Pattern = "s/\s*winfried\s*jakob/Winfried Jakob/i";

   PRX = prxparse(Pattern);

   Match_Result = prxmatch(PRX,Text);

   call prxchange(PRX,-1,Text);

   ... same code as before ...

run;
```

Example 2.1 String Replacement

```
PRX: s/\s*winfried\s*jakob/Winfried Jakob/i

Obs  1 No Match:  Ms. Winifred Jacob
Obs  2 No Match:  Winfred Jakob
Obs  3 No Match:  Wyn Fried Jakob
Obs  4 No Match:  Winfired Jakob
Obs  5 REPLACED:  Winfried Jakob
Obs  6 REPLACED:  Winfried Jakob
Obs  7 REPLACED:  Winfried Jakob
Obs  8 REPLACED:  Winfried Jakob
Obs  9 REPLACED:  Winfried Jakob
Obs 10 REPLACED:  Winfried Jakob
```

Example 3.1 String Extraction

```
Listing of Example 3 Data

Obs     Text

  1     153 First Street
  2     4000 Kanata Avenue North
  3     6789 64th Ave
  4     4 Moritz Road
  5     99 Borderline St.
  6     7493 Wilkes Place
  7     70 Sherring Court
  8     5 Goldridge Crescent
  9     7020 Yonge Street
 10     1 Buenaventura Pl. East
```

Example 3.1 String Extraction

```
data _null_;
   ... same code as before ...
   Pattern1 = "/\bavenue|\bave|\bcourt|\bcrt|";
   Pattern2 = "\bdrive|\bdr|\broad|\brd|\bstreet|\bst/i";
   PRX = prxparse(Pattern1 !! Pattern2);
   call prxsubstr(PRX, Text, position, length);
   if position ne 0 then do;
      match = substr(Text, position, length);
      putlog "Obs " _n_ 2.0 "  " match:$QUOTE. "found in " Text:$QUOTE. /;
   end;
   else if position eq 0 then do;
      putlog "Obs " _n_ 2.0 "  NO MATCH found in " Text:$QUOTE. /;
   end;
run;
```

Example 3.1 String Extraction

```
PRX: /\bavenue|\bave|\bcourt|\bcrt|
                \bdrive|\bdr|\broad|\brd|\bstreet|\bst/i
```

```
Obs  1 "Street" found in "153 First Street"
Obs  2 "Avenue" found in "4000 Kanata Avenue North"
Obs  3 "Ave" found in "6789 64th Ave"
Obs  4 "Road" found in "4 Moritz Road"
Obs  5 "St" found in "99 Borderline St."
Obs  6 NO MATCH found in "7493 Wilkes Place"
Obs  7 "Court" found in "70 Sherring Court"
Obs  8 NO MATCH found in "5 Goldridge Crescent"
Obs  9 "Street" found in "7020 Yonge Street"
Obs 10 NO MATCH found in "1 Buenaventura Pl. East"
```

Regular Expression Basics

Concatenation:

- The simplest form of a regular expression is simply a string of characters

Examples:

/Regular Expressions/
/Yonge Street/
/500 Dollars/
/xyz/
/1000/

Regular Expression Basics

Metacharacters:

- Some characters have a special meaning:
  { } [ ] ( ) ^ $. | * + ? \
- To find a string that contains any of those, you must prefix them with a backslash (\)

Example:

To find the string: “What is it?”
you must specify: /”What is it\?”/

Regular Expression Basics

More Metacharacters:

. the period matches any one character

\w a 'word'-like character, matches a-z, A-Z, 0-9, _

\d a 'digit' character, matches numbers 0 to 9

\s a 'space'-like character, matches any whitespace character, including the space and the tab

\b a word boundary, the position between a 'word' character and a 'non-word' character

Examples:

/\d\d:\d\d:\d\d/ matches the hh:mm:ss time format

/\bave/ does NOT match Buenaventura

Regular Expression Basics

Iterators:

*	matches 0 or more occurrences of preceding pattern
+	matches 1 or more occurrences of preceding pattern
?	matches 0 or 1 occurrences of preceding pattern
{k}	matches k occurrences of preceding pattern
{n,m}	matches at least n and at most m occurrences of preceding pattern

Example: /a{1,4}/ matches a, aa, aaa, aaaa

Regular Expression Basics

Alternation and Grouping:

- A vertical bar separates alternatives

Example: /Avenue|Ave|Boulevard|Blvd/

- Parentheses are used to define the scope and precedence of the operators

Example:

/H(ä|ae?)ndel/ matches Handel, Händel, and Haendel

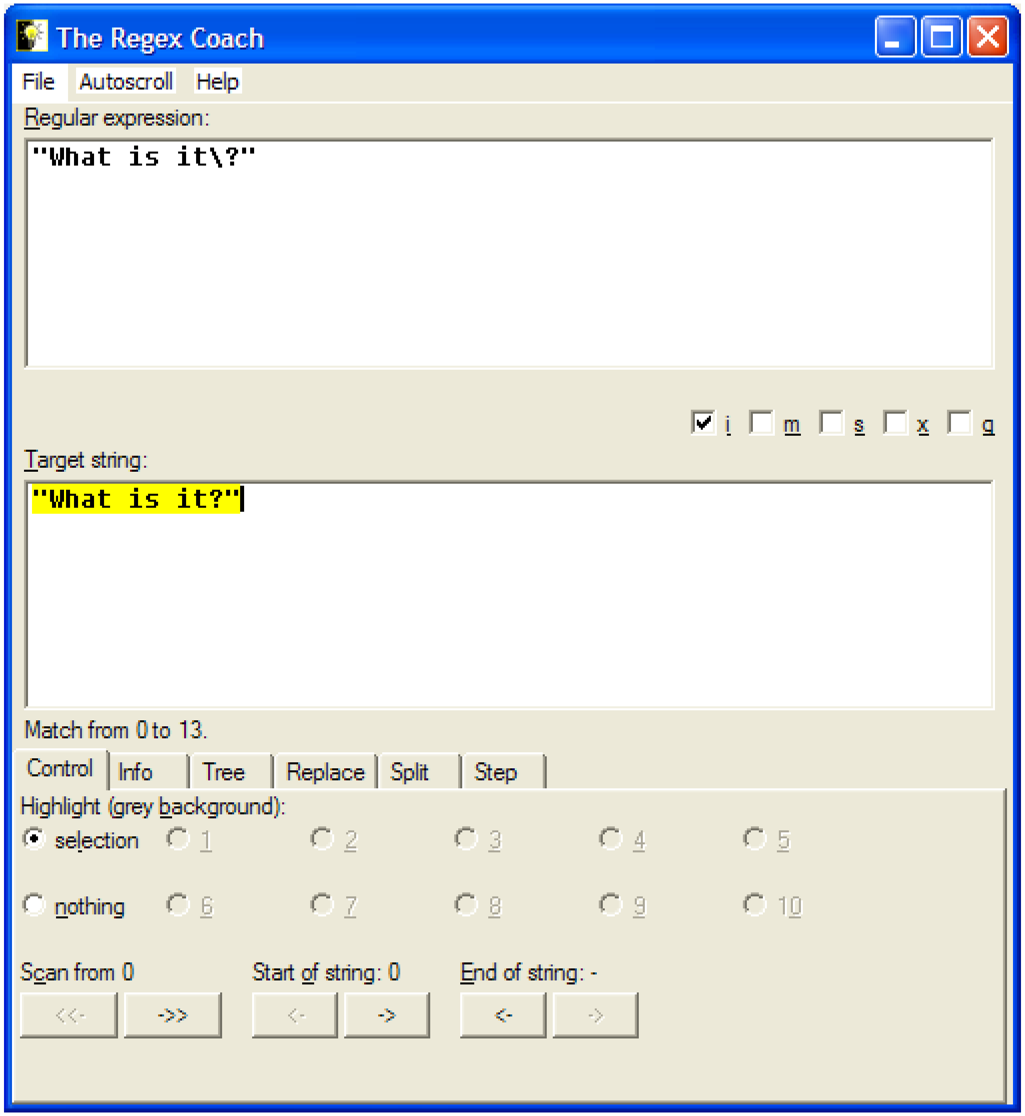
Regular Expression Basics

Character Classes:

- You can express single-character alternatives by using a character class.
- Character classes are denoted by square brackets, with the set of characters to be possibly matched inside.

Example: /[abc]ar/ matches aar, bar, and car

"The Regex Coach"[3]



Regular expression:

```
"What is it\?"
```

☑ i □ m □ s □ x □ g

Target string:

```
"What is it?"
```

Match from 0 to 13.

Highlight (grey background):
● selection ○ 1 ○ 2 ○ 3 ○ 4 ○ 5
○ nothing ○ 6 ○ 7 ○ 8 ○ 9 ○ 10

Scan from 0
Start of string: 0
End of string: -

- A great learning tool
- Free download
- Free use
- Highly recommended

A More Advanced Example: Problem

Check if a 6 character Postal Code follows these 3 rules:

- Postal Code must be in the format ANANAN. The first character must not be D, F, I, O, Q, U or W.
- Postal Code can also be a right-aligned mini postal code in the format: '   AA' (4 leading spaces)
- Postal Code can be 999999 for 'Unknown'.

A More Advanced Example: Solution

Construct 3 patterns and combine them as alternatives using the vertical bar:

- Pattern1 = "[ABCEGHJ-NPRSTVXYZ]\d[A-Z]\d[A-Z]\d";
- Pattern2 = "   [A-Z][A-Z]";
- Pattern3 = "999999";

PRX =

prxparse("/" !! Pattern1 !! "|" !! Pattern2 !! "|" !! Pattern3 !! "/");

A More Advanced Example: Result

```
Obs  1 MATCH in:  K2K2T1
Obs  2 No Match:  KKK2T1
Obs  3 No Match:  K2K211
Obs  4 No Match:  K+K2T1
Obs  5 No Match:     FL
Obs  6 MATCH in:      FL
Obs  7 No Match:       F
Obs  8 No Match:  99999
Obs  9 No Match:  999 99
Obs 10 MATCH in:  999999
```

How to Get Started

- Download this slide show from the OASUS website[4]
- Get SUGI 29 Article from Internet:

  David L. Cassell, Design Pathways, Corvallis, OR:
  "The Perks of PRX..."[2]

- Download and install "The Regex Coach"[3]
- Work through the article and try out your own ideas
- Learn by doing – and expect surprises!

Final Words of Wisdom...

Regular Expressions are extremely powerful, but they are not the best solution for every problem.

Learn enough to know when they are appropriate, and when they will simply cause more problems than they solve.

> **Some people, when confronted with**
> **a problem, think “I know, I'll use**
> **regular expressions.”**
> **Now they have two problems.**
>
> **Jamie Zawinski,**
> **in comp.lang.emacs**

References

1) Technical Support article: “Pattern Matching Using SAS Regular Expressions (RX) and Perl Regular Expressions (PRX)”

http://support.sas.com/91doc/getDoc/lrdict.hlp/a002288677.htm

2) Cassell, David L. (2004), “The Perks of PRX...”, SUGI Proceedings, 2003

http://www2.sas.com/proceedings/sugi29/129-29.pdf

3) “The Regex Coach” is available at: http://www.weitz.de/regex-coach/

4) Ottawa Area SAS Users Society website: http://www.oasus.ca

Acknowledgements

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.

The Perl language was designed by Larry Wall, and is available for public use under both the GNU GPL and the Perl Copyleft..

Other brand and product names are registered trademarks or trademarks of their respective companies.

About the presenter

Winfried Jakob

Senior Analyst, Data Quality

Canadian Institute for
Health Information

495 Richmond Road, Suite 600
Ottawa, Ontario K2A 4H6
Canada

Phone: +1 (613) 694-6993
Fax: +1 (613) 241-8120
Email: wjakob@cihi.ca

http://www.cihi.ca

Vildcard Side of SAS®
10 Minute SAS® 9 Tech Tip
presented by:
ied Jakob, Senior Analyst,
nadian Institute for Health I
Ottawa Congress Centre
Centre des congrès d'Ottawa