Sensitivity Training for PRXers
Kenneth W. Borowiak, PPD, Inc., Morrisville, NC

ABSTRACT

Any SAS® user who intends to use the Perl style regular expressions through the PRX family of functions and call routines should be required to go through sensitivity training. Is this because those who use PRX are mean and rude? Nay, but the regular expressions they write are case sensitive by default. This paper discusses the various ways to flip the case sensitivity switch for the entire or part of the regular expression, which can aid in making it more readable and succinct.

Keywords: case sensitivity, alternation, character classes, modifiers, mode modified span

INTRODUCTION

Any SAS user who intends to use the Perl style regular expressions through the PRX family of functions and call routines should be required to go through sensitivity training. Is this because those who use PRX are mean and rude? Nay, but the regular expressions they write are case sensitive by default. Before proceeding with an example, it will be assumed that the reader has at least some previous exposure to regular expressions. Now consider the two queries below in Figure 1 against the CHARACTERS data set for the abbreviation for ‘mister’ at the beginning of the free-text captured field NAME.

```
data characters;
  input name $40. ;
datalines;
  MR Bigglesworth
  Mini-mr bigglesworth
  Mr. Austin D. Powers
  dr evil
  mr bIgglesWorTH ;
proc print data=characters ;
  where prxmatch('/^MR/', name) ;       prxmatch('/^Mr/', name);
run;
```

<table>
<thead>
<tr>
<th>Obs</th>
<th>name</th>
<th>Obs</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MR Bigglesworth</td>
<td>3</td>
<td>Mr. Austin D. Powers</td>
</tr>
</tbody>
</table>

While the regular expressions ^MR and ^Mr specified as the first arguments in the PRXMATCH functions above look very similar, they generate different results. Both regexen (plural geek-speak for ‘regular expressions’) look for an upper case ‘M’ at the beginning of the field, as required by the anchor ^ . However, the first expression attempts to match an upper case ‘R’ in the second byte of the field while the second expression matches a lower case ‘r’. In addition, neither of the regexen match the fifth observation ‘mr bIgglesWorTH’ because of the lower case ‘m’ at the beginning of the field. This example demonstrates a very important concept: regular expressions are case sensitive by default. This paper discusses the various ways to flip the case-sensitivity switch for the entire or part of the regular expression, which can aid in making it more readable and succinct.

---

1 Cassell[2005] provides a nice introduction to the PRX functions and regular expressions.
CHARACTER CLASSES

One way to deal with case-sensitivity when needed is to use a character class. You specify a character class with a pair of brackets (i.e. [ ] ) and it allows any single character specified within to be matched. Suppose we are interested in displaying observations that begin with 'M' followed by either a lower or uppercase 'r'. Then we could write the regex as displayed in Figure 2.

![Figure 2](image)

Using a Character Class to Address Case-Sensitivity

/* Match 'M' followed by 'R' or 'r' at the beginning of the field */

```plaintext
proc print data=characters;
  where prxmatch('/^M[Rr]/', name);
run;
```

Obs            name
1     MR Bigglesworth
3     Mr. Austin D. Powers

When crafting a regex that needs to be case sensitive except at few locations with limited surrogates, character classes are a viable way to proceed. When using character classes, attention should be paid to the use of metacharacters, which are characters with a special meaning. The metacharacters that can be used within a character class do not necessarily have the same meaning when used outside of them. For example, ^M is a negated character class that allows any character except M to be matched. The metacharacter ^ in the first position inside the character class invokes the negation. The use of ^ in the first position of a regex, as in Figures 1 and 2, acts as an anchor that requires the match to begin at the beginning of the field.

ALTERNATION

Another way to deal with case-sensitivity when needed is to use alternation, which is specified with '|' and is analogous to the OR logical operator. Suppose we continue with the task presented in Figure 2, we could then write the regex as displayed in Figure 3A.

![Figure 3A](image)

Using Alternation to Address Case-Sensitivity

/* Match 'M' followed by 'R' or 'r' at the beginning of the field */

```plaintext
proc print data=characters;
  where prxmatch('/^M(R|r)/', name);
run;
```

Obs            name
1     MR Bigglesworth
3     Mr. Austin D. Powers

The regex matches the first and third observations from the CHARACTERS data set, as desired. Unfortunately, using parenthesis ( ) to group the alternatives causes a side effect in this simple pattern matching exercise. The parens cause the characters matched within to be
'captured' and held in a temporary variable, which could be extracted later using the PRXPOSN function\(^2\) or call routine. Since we are not interested in pulling out the matched characters in this example, there is no need to expend the resources needed to maintain the variable. For grouping only, we can use non-capturing parens specified with (?: ), as demonstrated below in Figure 3B.

**Figure 3B**
Using Alternation to Address Case-Sensitivity

```plaintext
/* Match 'M' followed by 'R' or 'r' at the beginning of the field */

proc print data=characters ;
  where prxmatch('/^M(?:R|r)/', name) ;
run ;
```

<table>
<thead>
<tr>
<th>Obs</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MR Bigglesworth</td>
</tr>
<tr>
<td>3</td>
<td>Mr. Austin D. Powers</td>
</tr>
</tbody>
</table>

Make the parens non-capturing by using (?:)

As with character classes, using alternation in a regular expression that needs to be case sensitive except at a limited number of locations is practical.

**i MODIFIER**

Let us now consider the task of matching records with the abbreviation for 'mister' at the beginning of the field while ignoring any differences in case. While we could write the regex using alternation or character classes, there is another option. Using the i modifier at the end of the regex makes the pattern search case-insensitive, as demonstrated below in Figure 4A.

**Figure 4A**
Case Insensitive Modifier i

```plaintext
/* Match 'M' followed by 'R' or 'r' at the beginning of the field */

proc print data=characters ;
  where prxmatch('/^Mr/i', name) ;
run ;
```

<table>
<thead>
<tr>
<th>Obs</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MR Bigglesworth</td>
</tr>
<tr>
<td>3</td>
<td>Mr. Austin D. Powers</td>
</tr>
<tr>
<td>5</td>
<td>mr bigglesWorTH</td>
</tr>
</tbody>
</table>

The i modifier is specified after the closing delimiter to make the search case insensitive

When the case-insensitive modifier is specified at the end of the regex it applies to the entire pattern. This changes the default behavior of the Perl style regular expressions, which are

---

\(^2\) PROC PRINT does provide a means to extract the submatch and create a new variable. In this example where we are just displaying results, you could extract the submatch and display with a single DATA or PROC SQL step.
case-sensitive. So an *isomorphic*\(^3\) regex could have been written as shown on the next page in Figure 4B.

![Figure 4B](image)

**Figure 4B**  
Case Insensitive Modifier i

/* Match 'M' followed by 'R' or 'r' at the beginning of the field */

```proc print data=characters;
  where prxmatch('/^mR/i', name);
run;
```

Obs   name
1    MR Bigglesworth
3    Mr. Austin D. Powers
5    mr bIgglesWorTH

This regex is isomorphic to the one in Figure 4A

Making the regex case-insensitive with the i modifier is more convenient, succinct, and arguably more readable relative to alternation and character classes when the pattern to match is long. Consider the example below in Figure 4C where a case-insensitive search for the word 'Bigglesworth' is desired.

![Figure 4C](image)

**Figure 4C**  
Case Insensitive Modifier i

```proc print data=characters;
  where prxmatch('/\bBigglesworth\b/i', name);
run;
```

Obs   name
1    MR Bigglesworth
2    Mini-mr bigglesworth
5    mr bIgglesWorTH

Using the i modifier can be much more succinct....

```proc print data=characters;
  where prxmatch('/\b[Bb][Ii][Gg]{2}[lL][eE][Ss][Ww][oO][Rr][tT][Hh]\b/', name);
run;
```

... than using character classes for dealing with case-insensitivity

While both regexen in Figure 4C generate the same result, you can see that needing to create a character class for every non-repeating alpha character (disregarding case) can be rather tedious.

**MODE MODIFIED SPAN**

The case when the entire pattern to be matched is to be case-insensitive has been examined. Let us now turn our attention to the case where only part of the expression is required to be

---

\(^3\)Objects that are isomorphic are structurally identical if you choose to ignore subtle differences that may arise from how they are defined.
case insensitive. Suppose we are interested in displaying observations that begin with a lower case 'b' at the beginning of a word followed by the string 'igglesworth', where the characters can be upper or lower case. The i modifier cannot help us here because it negates the ability to detect the lower case 'b' at the beginning of the word. This is where a mode modified span is useful. A case-insensitive mode modified span is specified by (?i:) within the delimiters of the regex and makes the subexpression within the parenthesis case-insensitive, as demonstrated below in Figure 5A.

```
proc print data=characters ;
  where prxmatch('/\bb(?i:iggLESwoRTh)\b/', name) ;
run ;
```

**Figure 5A**
Partial Case-Insensitivity with a Mode Modified Span

<table>
<thead>
<tr>
<th>Obs</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Mini-mr bigglesworth</td>
</tr>
<tr>
<td>5</td>
<td>mr bIgglesWorTH</td>
</tr>
</tbody>
</table>

Case-insensitivity is turned on for the submatch in parens

Note that when the i modifier is omitted after the closing delimiter in the expression the expression is case sensitive. This allows for the beginning lower case 'b' to be matched. The case-insensitive mode modifying span then overrides the case sensitive mode and allows the subexpression in the parens to be a case-insensitive match. The parens used in a mode modified span are non-capturing.

You can also create a case-sensitive mode modified span when the i modifier is specified, as shown below in Figure 5B.

```
proc print data=characters ;
  where prxmatch('/\b(?-i:b)iggLESwoRTh\b/i', name) ;
run ;
```

**Figure 5B**
Partial Case-Insensitivity with a Mode Modified Span

<table>
<thead>
<tr>
<th>Obs</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Mini-mr bigglesworth</td>
</tr>
<tr>
<td>5</td>
<td>mr bIgglesWorTH</td>
</tr>
</tbody>
</table>

Case-insensitivity is turned off for the submatch in parens

In this example, the mode modifying span is used to turn case sensitivity on in order to match the leading lower case 'b'. The syntax for a case-sensitive mode modifying span is to add a dash before the i in the non-capturing buffer.

---

4 You can think of a non-capturing buffer (?:) as a mode modifying span where the mode is unchanged.
**i MODE MODIFIER**

With the release of Version 9.2, another tool for handling case-sensitivity will be available. The *i mode modifier* (?) specified in the body of a regex overrides the default case-sensitive mode for the remainder of the regex, unless turned off with (?-i). The difference between the mode modifier and mode modified span is that the latter contains part of the regex within parens. The mode modifier does not enclose the remainder of the regex in parens. Continuing with the task presented in Figures 5A and 5B, a regex can be constructed using the i mode modifier as shown below in Figure 6.

![Figure 6](image)

**Figure 6**  
Partial Case-Insensitivity with a Mode Modifier

```sas
proc print data=characters ;
  where prxmatch('/\bb(?i)iggLESwoRTh\b/', name) ;
run ;
```

<table>
<thead>
<tr>
<th>Obs</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Mini-mr bigglesworth</td>
</tr>
<tr>
<td>5</td>
<td>mr bigglesWorTH</td>
</tr>
</tbody>
</table>

Case-insensitivity is turned on for the part of the regex that follows the mode modifier (?i)

This technique should not be used with the PRX functions and call routines prior to V9.2. In order to apply partial case-insensitivity in a regex in V9.1, use the techniques described in the earlier sections of the paper.

**CONCLUSION**

The Perl style regular expression that are available to the SAS user through the PRX functions and call routines are an invaluable tool for pattern matching, which may not require the alpha characters to match exactly. Regular expressions are case-sensitive, but the default setting can be turned off by the specifying the i modifier. Under certain conditions, the i modifier can be more convenient, succinct and readable vis-à-vis character classes and alternation. When you need part of the regex to be case-(?:in)?sensitive, modifying spans are handy.

You may have noticed there was hardly a mention of the relative efficiency amongst the various methods to handle case-insensitivity in regular expressions. The primary goal of the paper was to introduce the techniques. However, regardless of the method you employ to handle case-insensitivity, you are asking the regex engine to work harder and each of these techniques incurs a certain amount of overhead and resources. The internal workings of the regex engine are well beyond the scope of the paper. If you are interested in efficiency then you should consider benchmarking the various techniques against sample or hypothetical data.
REFERENCES


SAS OnLineDoc®


ACKNOWLEDGEMENTS

The author would like to thank Toby Dunn for his insightful comments in reviewing this paper. Jason Secosky provided valuable insight as to how PRX was implemented in SAS and what is forthcoming in future releases. The author would also like to thank his wife Jenni Borowiak for her tremendous support of my intellectual endeavors.

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 Perl Copyleft.

CONTACT INFORMATION

Your comments and questions are valued and encouraged.

Ken Borowiak
PPD, Inc.
3900 Paramount Parkway
Morrisville NC 27560

Ken.Borowiak@rtp.ppd.com
EvilPettingZoo97@aol.com