Regular Expressions in Create Lists (Revised for Sierra 3.4, April 2018)

1. Literal Characters and Metacharacters

Regular expressions are formed from combinations of literal characters and metacharacters. Literal characters are characters that represent themselves in a matching field. They include letters, numbers, the space, and most marks of punctuation. In Create Lists, the vertical bar ("|"), which is used as the subfield delimiter, is a literal character.

Metacharacters are those that perform some function within the regular expression. Some metacharacters work in combinations called "metasequences." The metacharacters and metasequences used in Create Lists are shown in the table below.

Regular expressions are invoked using the "matches" condition. As with all searches in Create Lists, regular expressions are case insensitive, with letters matching both upper and lower case.

Although the implementations of regular expressions in Millennium and Sierra are largely the same, there are significant differences. These will be discussed throughout this document.

Character Classes	
.	Period (or "dot"). Matches any single character.
[…]	User-defined character class. Matches any single character that is included in the class. Examples: **[aie]** the letter *a*, *e*, or *i* (upper or lower case) **[a-z]** matches any single letter (upper or lower case) **[a-z0-9]** matches any one letter or digit **[av-z]** matches any one of the letters *a*, *v*, *w*, *x*, *y*, or *z* **['"]** matches a single quote or a double quote **[- ,.]** matches a hyphen, space, comma or period (In the last example that the hyphen must come first or last so as not to be interpreted as a range indicator. Also note that the period character is treated as a literal within a character class.)
[^…]	Negated character class. Matches any single character that is *not* in the class. Examples: **[^]** any character that is not a space **[^0-9a-z]** any character that is not a digit or letter
Quantifiers	
+	Plus. Matches when the preceding character (or string of characters; see the section on grouping below) occurs 1 or more times. Example: **|a0+523** matches "|a0523", "|a00523", "|a000523", etc.
*****	Asterisk (or "star"). Matches when the preceding character (or string of characters) occurs 0 or more times. Examples: **|a0*523** matches "|a523", "|a0523", "|a00523", etc. **.*** matches any number of any characters

Richard V. Jackson, Huntington Library
rjackson@huntington.org

Quantifiers (continued)	
{ *min* **,** *max* **}** **{** *num* **}**	Matches when the preceding character (or string of characters) occurs at least *min* times, but no more than *max* times. When one number is given, the character (or string) must occur exactly *num* times. The largest number that may be used in a quantifier is 127 in Millennium, 255 in Sierra. Examples: **`[a-z]{5,8}`** matches a string of 5 to 8 letters **`[0-9]{4}`** matches a string of 4 numbers **`.{100,125}.{100,125}`** matches a string of 200 to 250 characters (These examples may match fields containing strings longer than the maximum value indicated, unless literal characters or other more specific subexpressions are included both before *and* after.)
? *(Sierra only)*	The question mark functions as a metacharacter ***only in Sierra***. It indicates that the preceding character (or string of characters) is optional. Examples: **`johnst?on`** matches "Johnson" or "Johnston" **`196[0-9][a-d]?`** matches "1960", "1961", … "1969", optionally followed by a letter "a", "b", "c", or "d" *In Millennium*, "**?**" is a literal character. Use "**`{0,1}`**" as an equivalent.
Grouping	
(…**)**	Allows a quantifier to apply to a string of characters. Example: **`Melvil(le){0,1} Dewey`** matches "Melville Dewey" or "Melvil Dewey" (the string "le" occurs 0 or 1 times) (In Sierra, this could be done with "**`Melvil(le)? Dewey`**".)
Position Indicators	
^	Beginning of the field position. Anchors what follows to the start of the field. (**^** must be the first character in the expression.) More information on p. 3.
$	End of field position. Anchors what precedes it to the end of the field. ($ must be the last character in the expression.) Examples: **`[^.]$`** matches when the last character in the field is not a period. (Note that the "**^**" within the brackets indicates a negated character class, not the beginning of the field.) **`^245..\|a.{0,5}$`** matches a 245 tag with 5 or fewer characters between |a and the end of the field
Backslash ("Escaping a metacharacter")	
**** *(Currently works in Millennium only)*	Applied before a metacharacter, causes it to be interpreted as a literal. (*Note: this does not work in Sierra. More on page 4.)* Examples: **`\.\.\.$`** matches 3 periods at the end of the field **`\$5[0-9]{2}\.[0-9]{2}`** matches "$"–"5"–2 numbers–decimal point–2 numbers (i.e. $500.00 to $599.99) *Sierra workaround:* place the character in square brackets: "**`[.][.][.]$`**"

Richard V. Jackson, Huntington Library
rjackson@huntington.org

2. How the System Stores MARC Data

Successful use of regular expressions, particularly the position indicators ("**^**" and "**$**"), requires understanding how your system stores MARC data. Some examples will illustrate:

- A MARC variable-length field includes a tag number, indicators, subfield delimiters, subfield codes, and data. The vertical bar ("|") is used as the subfield delimiter.

```
245  10  Nabi :|bpoema /|cpor José Carner.
```

```
245 10 |aNabi :|bpoema /|cpor José Carner.
```

Start of field position (**^**)
The first 3 characters of the field are the MARC tag number
The 4th and 5th characters are the two indicators
The first subfield delimiter and subfield code (including |a) occupy positions 6 and 7
Systems using Unicode store diacritics and special characters as a single character *
End of field position (**$**)

* *Millennium systems that do not store data in Unicode use curly brace codes, e.g., "***{226}e***" for "***é***", thus taking up an additional 5 character positions in the stored data.*

- When a particular MARC tag subfield is specified in a search statement, what constitutes the "field" varies between Millennium and Sierra. Millennium includes the initial delimiter; Sierra does not. For both systems, the "field" ends before the next delimiter, or at the end of the entire field.

```
700 1  |aHamilton, Alexander,|d1757-1804,|esigner
```

Millennium: Start of field position (**^**) End of field (**$**)
Sierra: Start of field position (**^**) End of field (**$**)

For example, to find added entry headings for persons born in the 1750s:

In Millennium, search:	**MARC Tag 700|d matches "^|d175"**
In Sierra, search:	**MARC Tag 700|d matches "^175"**

- The normalization rules that may apply to call numbers affect indexing only, not the way the call numbers appear to Create Lists.

```
05000|aPS3565.C57|bZ85 1994
```

- Non-MARC fields are stored without MARC tags, indicators, or subfield codes.

```
PATRON BARCODE     30006000411457
```

```
30006000411457
```

(**^**) (**$**)

Richard V. Jackson, Huntington Library
rjackson@huntington.org

3. Differences Between Millennium and Sierra

There are significant differences in behavior between Millennium and Sierra with regard to regular expression searches in Create Lists. Some things that work in Millennium will not work in Sierra, or must be done differently; the reverse is also true. In general, I have been able to find workarounds or alternate strategies to accomplish things in both systems.

The following summarizes the differences in behavior between the two systems, as far as my own testing has revealed. (Searches were done in Millennium 2011 1.6_5 and Sierra 3.4.0_10.)

The backslash ("\")

The backslash is a metacharacter used to "escape" other metacharacters, that is, to indicate that the next character should be treated as a literal character. Unfortunately, and contrary to the documentation, this does not work in Sierra in all releases at least through 3.4. (This is definitely a bug; it has now gone to Software Engineering.) Fortunately, there is a workaround: characters that are within character classes are treated as literal characters. For example:

To find fields that contain "**...**" (3 period characters)

\.\.\.	*Works in Millennium*
[.][.][.]	*Works in Sierra and Millennium*

To find "**[by]**" (e.g., in 245 |c)

\[by\]	*Works in Millennium*
[][by[]]	*Works in Sierra (at least in 3.4)*

The question mark ("?")

The question mark is a literal character in Millennium; however, in Sierra (and in most implementations of regular expressions) it is a metacharacter indicating that the preceding is optional. In Millennium, use "{0,1}" to indicate something is optional. For example:

To find either "**color**" or "**colour**"

colou?r	*Works in Sierra only*
colou{0,1}r	*Works in Millennium and Sierra*

To find fields ending in "**?**" (a literal question mark)

?$	*Works in Millennium; causes an error in Sierra*
\?$	*Also works in Millennium, but not in Sierra*
[?]$	*Works in both Millennium and Sierra*

MARC Tags 006, 007, 008

In Millennium, MARC fields 006, 007, and 008 can be searched directly in Create Lists. In Sierra, they cannot. Searches in Sierra using either of these forms:

```
BIBLIOGRAPHIC  MARC Tag 008   matches   "…"
BIBLIOGRAPHIC  MARC [field tag 'y']   matches   "^008…"
```

will either cause an error or fail to find any records. You must use the special fields defined for each component of the 006-008 (and the Leader) to search these fields. For example, to search for Date Types in 008 byte 06 with values "n", "q", or "u", you can search:

```
BIBLIOGRAPHIC  Dat Type  matches  "[nqu]"
```


Richard V. Jackson, Huntington Library
rjackson@huntington.org

POSIX Character Classes

Some POSIX character classes are supported in Sierra. These are special metasequences in the form "**[:…:]**". For example, "**[:alpha:]**" will match any ASCII letter, and "**[:digit:]**" will match any number. POSIX character classes *must always be used within regular character classes*; for example "**[[:digit:]x-]+**" will match a string containing digits, "x", "X", or "-", and "**[^[:alpha:]]**" will match any character that is not an ASCII letter.

More information about POSIX characters classes can easily be found on the Web. (See http://docs.racket-lang.org/guide/regexp-chars.html for example.) Not all of them work in Sierra, and some of them are pointless in the context of the kind of data being searched in Create Lists. For example, the case insensitivity of Create Lists renders "**[:upper:]**" (upper-case letters) and "**[:lower:]**" (lower-case letters) meaningless, and both "**[:blank:]**" and "**[:space:]**" are functionally identical to the space character.

The best use of a POSIX character class might be to find diacritics and special characters in Sierra. These can be found using "**[^[:ascii:]]**", which matches any character that is not an ASCII character.

Sierra's Differing Views of Variable-Length Fields

> *Note: The "disappearing |a" problem described in this section applies to earlier releases of Sierra (at least through 3.0). In more recent releases (3.4, possibly earlier), "|a" is consistently included whether you search by field group tag or MARC tag.*

In earlier versions of Sierra, the format of variable-length fields varies slightly, depending on whether the "field" being searched is defined with a field group tag or a MARC tag. When a field group tag is specified (e.g. "t" – TITLE matches …), the data appear as described on page 3:

```
24510|aA. Lincoln :|ba biography /|cRonald C. White, Jr.
```

However, when specifying a MARC tag in Sierra (by typing "!" and entering the MARC tag – e.g. 245), the first subfield delimiter and code disappear:

```
24510A. Lincoln :|ba biography /|cRonald C. White, Jr.
     ^
```

This strange behavior complicates searching with regular expressions in Sierra, especially when anchoring expressions to the beginning of the field or the first subfield code. For example, this search will find titles that begin with 3 numbers:

```
TITLE  matches  "^245.+|a[0-9]{3}"
```

However, this search will not (it works in Millennium though):

```
MARC Tag 245  matches  "|a[0-9]{3}"
```

Searching the 001 Field in Sierra

In Sierra (all releases), there is a variation in the way the 001 field is presented.

Field tag search: **OCLC# matches** …	The field looks like: **001 ocm12345678**
MARC tag search: **001 matches** …	The field looks like: **001ocm12345678**

The same thing is true of other 00X tags, although as mentioned above, the 006, 007, and 008 must be searched in Sierra using the "special fields" option.

Richard V. Jackson, Huntington Library
rjackson@huntington.org

Diacritics and Special Characters

In Millennium systems that store data in MARC-8, diacritics and special characters are coded as 3-digit numbers in curly braces. (See IGR # 105473 or # 101353 for a list of them.) To match a particular one, such as the degree sign ("º"), enter its code as: "**\{493\}**". To match *any* diacritic or special character, search for: "**\{[0-9]{3}\}**".

In Millennium systems that store data in Unicode, diacritics and special characters are stored with curly braces enclosing "u" and 4 hexadecimal digits, e.g. "**{u00AE}**". To match *any* of these characters, search "**\{u[0-9A-F]{4}\}**".

Sierra systems store data in Unicode, but you cannot use a regular expression within a curly-brace code as above. You may search for a *particular* special character by pasting it directly in the expression, or by entering its exact Unicode curly-brace code:

PUB INFO matches "|c[c©]200[1-5]"	dates preceded by either "c" or "©"
TITLE matches "sh[o{u014d}]gun"	finds "Shogun" or "Shōgun"

Note that the Unicode code (e.g., "{u014d}") is interpreted as the actual character ("ō") prior to evaluation by the regex engine. Thus the curly braces are not treated as regex metacharacters in this situation. (Thanks to Lloyd Chittenden for calling my attention to this.)

To find *any* diacritic or special character in Sierra, search: "**[^[:ascii:]]**"

*The Metasequence "**[^|]**"*

The most straightforward way to match the *content* of any subfield is with the expression: "**[^|]+**" (one or more characters that are not a subfield delimiter). For example, this search should match 245 fields that contain no further subfields after subfield |a:

```
TITLE  matches  "^245.+|a[^|]+$"
```

(The content of subfield |a extends all the way to the end of the field.)

Unfortunately, *this does not work in Sierra*. Although the documentation mentions this particular expression, in all releases of Sierra the expression "**[^|]**" fails to match anything.

There are no easy workarounds. Probably the best approach is to use a positive character class rather than a negated one. Here is an imperfect workaround for the above example:

```
TITLE  matches  "^245.+|a[ -{}~]+$"
```

The character class "**[-{}~]**" will match any ASCII character in the range from the space character (hex 20) to left brace "{" (hex 7B), plus right brace "}" (hex 7D) and the spacing tilde "~" (hex 7E). This excludes the subfield delimiter "|" (hex 7C). Although this character class will work in many situations, in this example, it will fail if 245|a contains any diacritics or special characters. As explained above, such characters can be matched in Sierra with "**[^[:ascii:]]**", a negated character class that matches non-ASCII characters. To combine this negated character class with a positive character class, it is necessary to group them like this:

```
TITLE  matches  "^245.+|a([ -{}~]*[^[:ascii:]]*)+$"
```

So … to match one or more characters that are not subfield delimiters:

Use this in Millennium:	**[^|]+**
Use this in Sierra:	**([-{}~]*[^[:ascii:]]*)+**

This problem was reported and has now gone to Software Engineering.

Richard V. Jackson, Huntington Library
rjackson@huntington.org

4. Examples of Regular Expressions in Create Lists

Except where indicated, searches in these examples should work in both Millennium and Sierra.

Example 1: Use of the "dot" metacharacter

Problem: You need to limit a search to titles published in the United States. Unfortunately, the fixed-length field COUNTRY uses separate codes for each of the 50 states, D.C., etc.

Solution:

COUNTRY matches "..u"

All U.S. codes have "u" as the third character. With this expression, the first two characters can be anything, but the third character must be "u", matching all U.S. codes.

Example 2: Character classes

Problem: Find notes containing the phrase "*gray* [or *grey*] *wolf* [or *wolves*]."

Solution:

NOTE matches "gr[ae]y wol[fv]"

Note that a character class (without a quantifier) must match one and only one character.

Example 3: Negated character classes

Problem: Look for missing (or invalid) subfield codes, such as the ones in these fields:

```
245 10 |aLove for love|[microform] :|ba comedy /|cby William Congreve.
                      ^
650  0 |aUnited States|xHistory|Civil War, 1861-1865|vAnecdotes.
                               ^
```

Solution:

MARC Tag 245 matches "|[^abcfghknps6]"

MARC Tag 650 matches "|[^avx-z]"

Negated character classes work well for finding invalid characters. Valid subfield codes for the 245 tag, for example, are a, b, c, f, g, h, k, n, p, s and 6. These expressions match a subfield delimiter ("|") followed by a character that is not a valid code.

Example 4: Use of "dot-star"

Problem: Look for non-repeatable subfield codes that are repeated, such as these:

```
245 10 Deception :|ba novel /|by Philip Roth.
245 10 California :|ca history /|cAndrew F. Rolle.
```

Solution:

MARC Tag 245 matches "|b.*|b"
OR MARC Tag 245 matches "|c.*|c"

The construction "dot-star" (`.*`) is often used as a placeholder for "any number of any characters" between two more specific sub-expressions.

Richard V. Jackson, Huntington Library
rjackson@huntington.org

Example 5: More with “dot-star”

Problem: Create a bibliography of anything relating to 18th century France.

Solution:

```
     SUBJECT  matches  "france.*18th cent"
  OR SUBJECT  matches  "france.*17[0-9][0-9]"
```

This search will match all these headings and more:

```
651  0 France|xHistory|yRevolution, 1789-1799|xArt and the revolution
600 00 Louis|bXIV,|cKing of France,|d1638-1715|xArt collections
651  0 France|xIntellectual life|y18th century
650  0 Books and reading|zFrance|xHistory|y18th century
650  0 Architecture|zFrance|zParis|y18th century
651  0 France|xPolitics and government|y1789-1815
650  0 Printing|zFrance|xHistory|y18th century
650  0 Republicanism|zFrance|xHistory|y18th century
650  0 Individualism|xSocial aspects|zFrance|xHistory|y18th century
650  0 Ethnopsychology|zFrance|xHistory|y18th century
651  0 Paris (France)|xHistory|y1799-1815
651  0 Lyon (France)|xHistory|xSiege, 1793
651  0 Paris (France)|xSocial life and customs|y18th century
600 10 Douglas, Frances,|cLady,|d1750-1817        [false drop]
```

Example 6: The {min,max} quantifier

Problem: Find words that may be spelled differently, or phrases with optional words.

Solution:

```
ORDER NOTE  matches  "cancel{1,2}ed"

     TITLE  matches  "Thomas (Alva ){0,1}Edison"
```

Alternate solution in Sierra for the second search:

```
     TITLE  matches  "Thomas (Alva )?Edison"
```

Example 7: Treating a metacharacter as a literal

Problem: Find ISBNs in bib records that contain a price in dollars in subfield |c. The price should be less than $100.

Solution *in Millennium*:

```
MARC Tag 020|c  matches  "\$[1-9]{0,1}[0-9]\."
```

Solution *in Sierra*:

```
MARC Tag 020|c  matches  "[$][1-9]?[0-9][.]"
```

This matches the dollar sign, followed by a number (1-9), then a second number (0-9), then a decimal point. To account for values between $0.01 and $9.99, the first number is made optional.

Both the dollar sign and the decimal point (period) are normally metacharacters. To search for them as literal characters, precede them with a backslash (“\”) in Millennium. Until this is fixed in Sierra, you must place a metacharacter in a character class for it to be treated as a literal.

Richard V. Jackson, Huntington Library
rjackson@huntington.org

Example 8: Position indicators

Problem: Find subject headings with second indicators that are not 0 (Lib. of Congress), 5 (Natl. Lib. of Canada), or 7. Also find those with 2nd indicator 7, where the last subfield is not |2.

Solution:

```
        SUBJECT  matches  "^6...[^057]"
OR      SUBJECT  matches  "^6...7.*|[^2][^|]+$"
```

The first search statement matches a 6 at the start of the field, followed by any 3 characters (for the rest of the tag number and the first indicator), followed by a character that is not 0, 5, or 7. The second search statement matches a 6 at the beginning, a 7 as the second indicator, any number of any characters, then a subfield delimiter with a subfield code other than 2, followed by one or more characters that are not another subfield delimiter, followed by the end of the field. The last part ensures that the subfield code that is not |2 is the last subfield in the field. *Note: This second statement doesn't work in Sierra; see discussion (and a workaround) on page 6.*

Example 9

Problem: Find system control numbers in MARC tag 035 that are exactly 4 digits long in order to use Global Update to insert 2 leading zeroes.

Solution:

```
        MARC  matches  "^035..|a[0-9]{4}$"
OR      MARC  matches  "^035..|a[0-9]{4}[^0-9]"
```

This search matches |a, followed by exactly 4 digits, followed by either the end of the field or a character that is not a digit. Global Update can then be used to insert "00" at the start of the field. (Note: The field group tag associated with MARC tag 035 might be different in your library.)

Example 10

Problem: Find records where the title proper (MARC tag 245 |a) is longer than 300 characters.

Solution:

```
MARC Tag 245|a  matches  ".{100}.{100}.{100}"
```

The maximum value for any quantifier is 127 in Millennium, 255 in Sierra, so the expression "`.{300}`" won't work. Instead, use multiple quantified subexpressions to reach the desired maximum. This expression finds subfields that *contain* 300 characters; they can be longer.

Example 11

Problem: MARC tag 040|b contains a language code indicating the language of cataloging (not the same thing as the language of the resource). For example, a catalog record originating from *Die deutsche Nationalbibliothek* might have: **040|aGWDNB|bger|erakwb|cGWDNB** …

You want to survey all cataloging records in which 040|b is a language other than English (eng). Finding the *absence* of a string of characters can be difficult, given the lack of a "NOT" operator or a "Does not match" condition. Doing it with regular expressions requires 3 search statements:

```
        MARC Tag 040  matches  "|b[^e]"
OR      MARC Tag 040  matches  "|be[^n]"
OR      MARC Tag 040  matches  "|ben[^g]"
```


Richard V. Jackson, Huntington Library
rjackson@huntington.org

Example 12: Using the fixed-length fields

Problem: Limit a search to titles published *outside* the United States.

Solution:

```
COUNTRY  matches  "^..[^u]"
```

These match records where the last character of the fixed-length field COUNTRY is not "u". Here is another example, which matches Country codes for Canada outside of Ontario and Québec.:

```
COUNTRY  matches  "[^oq].c"
```

Sierra's Country Code Quirk

There is an odd quirk in Sierra regarding Country codes that is worth noting here. Generally, fixed-length fields such as Location are padded out to their full length with trailing spaces. The 3-character Country code in Sierra seems to be an exception; trailing spaces are absent. (The same thing applies to the COUNTRY special field [008/15-17].) Thus, the code for France, for example, is stored as "**fr**<space>" in Millennium, but as "**fr**" in Sierra. Therefore, the following search, intended to match places in the United States:

```
COUNTRY  matches  "u$"
```
[Country ends in "u"]

in Sierra will also match codes such as "hu" (Hungary), "lu" (Luxembourg), and several others. Likewise, this search for places outside the United States:

```
COUNTRY  matches  "[^u]$"
```
[Country ends in something other than "u"]

in Sierra will fail to match codes such as "hu " and "lu ".

When searching the Country field in Sierra with regular expressions, it is better to count characters from the start of the field as described above. (This too is in Software Engineering.)

Example 13

Problem: Some titles have incorrect filing indicators, such as:

```
245 0 4 |aGrand Tour :|bthe lure of Italy in the eighteenth century
245 0 3 |aA letter from a Gentleman of the City of New-York to another
```

(The first title is indexed as "d tour the lure of italy…", the second as "etter from a gentleman…")

Solution:

```
      TITLE  matches  "^245.2.*|a.[^ '"]"
OR    TITLE  matches  "^245.3.*|a..[^ '"]"
OR    TITLE  matches  "^245.4.*|a...[^ '"]"
OR    TITLE  matches  "^245.5.*|a....[^ '"]"
```
[etc.]

These expressions depend on the fact that for a filing indicator *n*, the *n*th character following "|a" should normally be a space, apostrophe/single quote, or double quote. The above expressions match when the *n*th character is not one of these characters. The "**.***" following the second indicator is to account for any titles in which |a is not the first subfield, such as those that have a subfield |6 with a link to an 880 tag.

This will match correctly coded Arabic titles such as "245 13 |aal-Kūfah…". Excluding them with: "**^245.3.*|a..[^ '"-]**" works, but will fail to find, say, "245 13 |aIn-laws and outlaws…"

Richard V. Jackson, Huntington Library
rjackson@huntington.org

Example 14

Problem: Find subjects headings that have a second indicator 4, including those in MARC tags 600, 610, 611, 630, 650, and 651, but not including 655 or 690.

Solution:

```
SUBJECT  matches  "^6[0135][01].4"
```

Use "^" to anchor the expression to the beginning of the field. In this expression, 655s are excluded because the 3rd digit can only be 0 or 1. 690s are excluded because the 2nd digit cannot be 9. The first indicator can be any character, but the second indicator must be 4.

Example 15

Problem: In MARC tag 856 (Electronic location and access), a clickable "linking" text may be generated in the OPAC from |z (Public note) or |3 (Materials specified). Find records with 856s that contain neither |3 nor |z.

Solution *in Millennium*:

```
MARC  matches  "^856..(|[^3z][^|]*)+$"
```

Use both the start and end of field position indicators, and account for the tag number and indicators at the beginning. The parentheses group together the characters of a single subfield, here specified as a subfield delimiter ("|"), followed by a subfield code that is not "3" or "z", followed by any number of characters that are not "|". The "+" indicates one or more such subfields occurring until the end of the field is reached.

It might seem simpler to use:

```
    MARC Tag 856  at least one field does not have  "|z"
AND MARC Tag 856  at least one field does not have  "|3"
```

However, a single record may have multiple 856s. The above search may find records where one 856 lacks "|z" and another lacks "|3." Only the regular expression guarantees finding instances of a single field that lack both subfields.

A solution *in Sierra* that follows the same pattern is even more complex. As noted above, the expression "**[^|]**" does not work in Sierra. However, the following, based on the workaround described on p. 6, will work:

```
MARC matches "^856..(|[^3z]([ -{}~]*[^[:ascii:]]*)*)+$"
```

Fortunately, Sierra's Enhanced Query Builder (release 2.2 +) provides a much simpler solution, without need for a regular expression:

```
MARC Tag 856
       at least one field doesn't have  "|z"
     AND
       at least one field doesn't have  "|3"
```

The Enhanced Query Builder allows multiple terms to be applied to one target field, which ensures that all conditions will be met *in the same instance* of the field. In this case, the record will be retrieved only if at least one 856 lacks both "|z" and "|3".

Richard V. Jackson, Huntington Library
rjackson@huntington.org

Example 16

Problem: Find records that contain a bad code in the fixed-length field MAT TYPE (aka Bcode2).

Solution:

```
MAT TYPE  matches  "[^ac-gijkmoprt]"
```

The user manuals for both Millennium and Sierra state that regular expressions can only be used on certain fixed-length fields, specifically, those that are longer than a single character. *This is not true.* Expressions such as the above work just fine on the single-character MAT TYPE field.

A similar search can be used to exclude suppressed records. For example, if BCODE3 uses **l**, **n**, and **p** as suppression codes, including this search will omit suppressed records from the results:

```
… AND  BCODE3  matches  "[^lnp]"
```

Example 17

Problem: Find books where the number of pages given in 300 |a is 25 or fewer.

Solution:

```
     300|a  matches  "[^0-9][1-9] p[.]"
OR   300|a  matches  "[^0-9]1[0-9] p[.]"
OR   300|a  matches  "[^0-9]2[0-5] p[.]"
```

Separate search statements are needed to find books with 1-9, 10-19, and 20-25 pages. Including "[^0-9]" at the beginning of each expression ensures that values such as "623 p." are not matched, while still permitting matches on values such as "vi, 23 p." Unfortunately, this search will also match, for example, "123 p., 16 p. of plates", and will fail to match "[8] p." Also, to account for RDA-style records, you may need additional statements with "page" instead of "p[.]".

Example 18

Problem: Find bib records that have two (or more) call number fields. The only characteristic that reliably distinguishes the call numbers is that one is always longer than the other.

Solution:

```
      (   CALL #  matches  "|a.{6}$"
AND       CALL #  matches  "|a.{7}"    )
OR    (   CALL #  matches  "|a.{7}$"
AND       CALL #  matches  "|a.{8}"    )
OR    (   CALL #  matches  "|a.{8}$"
AND       CALL #  matches  "|a.{9}"    )
OR    [etc.]
```

This search assumes that the shortest possible call number contains 6 characters following "|a". The first pair of search statements matches records where one call number has exactly 6 characters and another call number in the same record contains at least 7 characters. (Note the presence or absence of the end of field indicator ($).) The second pair of search statements increments these lengths by one, finding one call number with exactly 7 characters and another with 8 or more. You will need to add enough pairs of statements to reach the maximum likely length for the call number. The parentheses shown grouping each pair of statements above are not necessary, but are included here for clarity. *(Problems like this are better solved using SQL.)*

5. Metacharacters That Do *Not* Work in Create Lists

Certain metacharacters and metasequences that are available in other implementations of regular expressions do not work in Create Lists. Some of these are listed here, along with some possible workarounds.

(…|…)	Used to indicate two or more alternative strings of characters, for example "**(donation|gift)**". In Create Lists, "**|**" is a literal (the subfield delimiter). *Workaround:* There is no workaround within a single regular expression; however, you may be able to accomplish the same thing with multiple search statements linked with Boolean OR.
(…)…\1	Back references. Allows a string of text to be "captured" so the same text may be matched on again later in the same expression (or may be inserted in the replacement text). Back references are not available in Create Lists. There is no workaround. However, back references are primarily useful in match-and-replace operations, and would have limited use in Create Lists.
\w, **\d**, **\b**, **\s**, **\<**, **\>**, etc.	Special character classes, positions, etc., such as "**\w**" (any alphanumeric word character) or "**\b**" (a word boundary). None of these work in Create Lists. The backslash is used only to change a metacharacter into a literal character (in Millennium). (Backslashing a literal character simply results in the same literal character.) *Workarounds:* Most of these character classes are easily rendered using other metacharacters. For example, "**\d**" (any digit) can be rendered as "**[0-9]**". Using word boundary positions is problematic in MARC format anyway, because of the use of letters for subfield codes.

6. Further information

The User Manuals — IGR page # 100672 for Millennium, or Sierra Guide > Creating Lists (Review Files) > Using Boolean Searching — give only a little information on the "matches" condition in Create Lists, and not all of it is accurate. There is a wealth of information on the Web and elsewhere on regular expressions; however, much of it is confusing, misleading, or not applicable to Create Lists. Most of what I have learned has been through a combination of reading and experimentation. I did find the following book to be quite useful:

Friedl, Jeffrey E. F. *Mastering Regular Expressions*, 3rd edition. Sebastopol, CA: O'Reilly, 2006. 515 pp.

This book, also from O'Reilly, looks easier to digest:

Fitzgerald, Michael. Introducing Regular Expressions, 1st edition. Sebastopol, CA : O'Reilly, 2012. 136 pp.

Finally, here is a website for testing regular expressions: https://www.regexpal.com/

Richard V. Jackson, Huntington Library
rjackson@huntington.org