# Regular Expressions in Create Lists (Revised for Sierra 5.2, March 2021)

Richard V. Jackson, The Huntington Library

Regular expressions, or regexes, are a powerful tool for matching patterns of characters rather than specific values. In Create Lists, regexes are invoked with the "matches" or "matches (exact)" condition, the latter being case-sensitive. Regexes use combinations of literal characters and certain special characters, called metacharacters. Rather than matching literal values, metacharacters perform some function within the regular expression. These characters and their functions are described below.

With Millennium and the earlier releases of Sierra, searches with regular expressions in Create Lists are/were based on the "egrep" standard. With Sierra release 4.1, Innovative switched to the POSIX implementation of regular expressions, which comes in two "flavors": Basic Regular Expressions (BRE) and Extended Regular Expressions (ERE). BRE is the default but is somewhat limited in scope. ERE is more powerful and is similar to the "egrep"-based regexes used earlier. To switch from the default BRE to an ERE, you must enter the string "**(?e)**" at the beginning of the expression.

Example of a BRE query:

| | | | |
|---|---|---|---|
| **BIBLIOGRAPHIC** | **AUTHOR** | **matches** | **\|ap.*, edm[ou]nd** |

(matches authors with last name starting with "P" and first name "Edmond," "Edmund", etc.)

Example of an ERE query:

| | | | |
|---|---|---|---|
| **BIBLIOGRAPHIC** | **AUTHOR** | **matches** | **(?e)\|d(approximately )?16[0-3][0-9]** |

(matches authors born 1600-1639, possibly with approximate birthdates)

The information in this handout was tested under Sierra release 5.2. Except where noted, most of this information should apply to releases back to Sierra 4.1.

## Metacharacters for Basic and Extended Regular Expressions

The following metacharacters may be used with both BRE or ERE:

| *Character Classes* | |
|---|---|
| **.** | Period (or "dot"). Matches any single character. |
| **[…]** | User-defined character class. Matches any single character that is included in the class. Examples:<br>**[aie]** the letter *a*, *e*, or *i* (upper or lower case)<br>**[a-z]** matches any single letter (upper or lower case)<br>**[a-z0-9]** matches any one letter or digit<br>**[av-z]** matches any one of the letters *a*, *v*, *w*, *x*, *y*, or *z*<br>**['"]** matches a single quote or a double quote<br>**[- ,.]** matches a hyphen, space, comma or period<br>(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:<br>**[^ ]** any character that is not a space<br>**[^0-9a-z]** any character that is not a digit or letter |
| *Quantifiers* | |
| ***** | Asterisk (or "star"). Matches when the preceding character or character class occurs 0 or more times. Examples:<br>**\|a[0-9]* *repro** matches "\|a", any number of digits, any number of spaces, and "repro" (i.e. "\|a5678 repro", "\|a123repro", "\|arepro", etc.)<br>**.*** matches any number of any characters<br>(The asterisk is the only quantifier available under BRE; use ERE for others such as **+**, **?**, and **{**…**}**.) |
| *Position Indicators* | |
| **^** | Circumflex (or caret). Indicates the start of field position. Anchors what follows to the beginning of the field. (**^** must be the first character in the expression.)<br>**^[^3]** matches fields where the first character is not 3<br>(Note: the first "^" indicates the start of field; the second one indicates a negated character class.) |
| **$** | End of field position. Anchors what precedes it to the end of the field. ($ must be the last character in the expression.) Examples:<br>**[?!"]$** matches when the field ends with ?, !, or "<br>**[^.]$** matches when the last character in the field is not a period. |
| *Backslash ("Escaping a metacharacter")* | |
| **** | Applied before a metacharacter, causes it to be interpreted as a literal. Examples:<br>**\.\.\.$** matches 3 periods at the end of the field<br>**\$[5-9][0-9]\.[0-9][0-9]** matches "$" – "5" to "9" – another number – decimal point – 2 more numbers (i.e. $50.00 to $99.99) |


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

The following metacharacters are available only under ERE; the regex must begin with "**(?e)**":

| *Quantifiers (continued)* | |
|---|---|
| **+** | Plus. Matches when the preceding character or character class occurs 1 or more times. Example:<br>**\|a0+523** matches "\|a0523", "\|a00523", "\|a000523", etc. |
| **?** | Question mark. Indicates that the preceding character or character class is optional (occurs 0 or 1 times). Examples:<br>**johnst?on** matches "Johnson" or "Johnston"<br>**196[0-9][a-d]?** matches "1960", "1961", … "1969", optionally followed by a letter "a", "b", "c", or "d" |
| **{***min***,** *max***}**<br>**{***num***}** | Matches when the preceding character or character class occurs at least *min* times, but no more than *max* times. When one number is given, it must occur exactly *num* times. The largest maximum that may be used is 255. Examples:<br>**[a-z]{5,8}** matches a string of 5 to 8 letters<br>**[0-9]{14}** matches a string of 14 numbers<br>(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.)<br>It is possible to use curly-brace quantifiers under BRE, but you have to "escape" each brace. For example:<br>**[0-9]\{14\}** |
| *Grouping* | |
| **(**…**)** | Parentheses. Allow a string of characters to be treated as a unit, for example to apply a quantifier:<br>**melvil(le)? dewey** matches "Melville Dewey" or "Melvil Dewey"<br>As with curly braces, it is possible to use grouping under BRE, but you have to "escape" each parentheses. The example would have to be entered as:<br>**Melvil\(le\)\{0,1\} dewey**<br>More on this in the section on back references on the next page. |
| *Alternation* | |
| **…\|…** | Allows 2 or more alternate strings or expressions to satisfy the match:<br>**fre\|spa\|ita** matches "fre" or "spa" or "ita" (as in the LANG field)<br>Usually, you will need to group the alternation in parentheses to separate it from other parts of the expression that do not participate in the alternation:<br>**(gr[ae]y\|timber\|lone) wol(f\|ves)**<br>Note: In BRE, the vertical bar is a literal character (the subfield delimiter); in ERE it is a metacharacter. When switching to ERE, remember to "escape" the vertical bar ("**\|**") to enter a subfield delimiter:<br>**\|a(london\|oxford).* :\|b(published by\|printed for)** |


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

# Features That Only Work with Basic Regular Expressions

## *Word Boundaries*

A query on the string “cat” will match not only the word “cat” but also “cattle”, “concatenate”, etc. Occasionally, it may be helpful to require that a character occur at the beginning of a word or at the end of a word. When using BRE, it is possible to do this using these metasequences:

**\<** = start of word position; the point where a non-word character is followed by a word character
**\>** = end of word position; the point where a word character is followed by a non-word character

These are analogous to “**^**” (start of field position) and “**$**” (end of field position), except they apply to words. “Words” (at least in this regex implementation) can include ASCII letters, numbers, and the underscore. All other characters (as far as I can tell) are non-word characters.

Thus, a search for “**\<cat\>**” will match if the “c” is at the beginning of a word and the “t” is at the end of a word. In plain language, this will match the word “cat”. However, there are two issues that can lead to unexpected results, making searches like this less effective:

(1) When a word occurs at the beginning of a MARC subfield, the subfield code will be treated as part of the word. So a query on “TITLE matches \<cat\>” will also find, for example:

```
245 10 |a … /|cat the suggestion, and under the superintendence of …
```

(2) Letters with diacritics unfortunately are considered non-word characters, so a search for “**\<cat\>**” will also match “catálogo”, “catástrofe”, and “Bécat”.

Because this feature only works under BRE, which means having to forego other functionality, its use may be limited. Advanced word searching may be a better option for finding words and phrases.

## *Back References*

Back references are a way to capture and store a string of characters found in a field and then refer back to it later in the expression. They were never available in Create Lists under the previous implementation of the “egrep” standard, but they can be used now under POSIX. Like word boundaries, back references will only work under BRE.

There are two parts to a back reference. The first part uses grouping to capture and remember a string of characters. Because they are used under BRE, the parentheses must be “escaped.”

**\(|[abc]\)** captures a subfield delimiter followed by subfield code *a*, *b*, or *c*

The second part of the back reference refers back to what was captured using **\1**. If there is more than one grouping, you can refer to subsequent groupings with **\2**, **\3**, … **\9**. In Create Lists, it seems unlikely you would need more than one.

This example finds titles in MARC tag 245 in which a captured occurrence of “|a”, “|b”, or “|c” appears again later in the field:

| BIBLIOGRAPHIC | TITLE | matches | ^245.*\(\|[abc]\).*\1 |
|---|---|---|---|

This matches fields such as:

```
245 10 |aEpigrammatum Graecorum/|annotationibus Ioannis …
245 10 |a1876 :|ba novel /|by Gore Vidal.
245 10 |aCalifornia:|ca history /|cAndrew F. Rolle.
```

Back references are mainly used in match-and-replace operations, and their use in Create Lists is limited to finding a string of characters that is repeated in the same field. Another example is given on page 13 in the “Examples” section.


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

## Case-Sensitive Queries

Searches in Create Lists are normally case-insensitive, with ASCII letters a-z matching both upper- and lower-case letters. In Sierra release 5.0, new conditions in Create Lists were added that permit case-sensitive searching. There are now four such conditions:

| | | | |
|---|---|---|---|
| has (exact) | *[type* "**x**" *to enter]* | starts with (exact) | *[type* "**s**" *to enter]* |
| matches (exact) | *[type* "**j**" *to enter]* | ends with (exact) | *[type* "**z**" *to enter]* |

When using "matches (exact)," a character class such as "**[a-z]**" will match only lower-case case letters. Use "**[a-zA-Z]**" to match any letter. Likewise, the negated character class "**[^a-z]**" will still match upper-case letters.

Note that letters with diacritics are always case-sensitive. For example, "**é**" will not match "**É**" when using either "matches" or "matches (exact)."

## Entering characters with Unicode notation

Sierra stores text data in Unicode. Any character may be entered using Unicode notation, both in record editing and in Create Lists queries. The notation follows this format: **{u**####**}**, where "####" is the 4-digit hexadecimal code for the character. This can be useful for entering symbols and characters with diacritics that don't appear on the standard keyboard. For example, **{u00f8}** = "ø" and **{u2113}** = "ℓ". The system recognizes characters entered this way as Unicode notation and converts the string to the actual character.

This can cause some confusion when using regular expressions in Create Lists, since the curly braces are metacharacters used in numerical quantifiers. The Sierra manual states that you have to "escape" the braces (e.g. "**\{u00A8\}**"), *but this is incorrect*; you will get an error if you do this.* Unicode notation may be used in regexes as you would anywhere else. The system converts the notation to the actual character *prior to processing by the regex engine*, so the braces are not interpreted as metacharacters. This also means you cannot use metacharacters within the Unicode notation. For example, searching for "**{u03..}**" (or "**\{u03..\}**") will not work. However, you may use a character class to search for a range of Unicode values. For example:

**[{u0370}-{u03ff}]** will match any character in the Greek/Coptic character set

The website https://unicode-table.com/ provides a browsable and searchable table of Unicode characters and their hexadecimal codes.

* Innopac and non-Unicode Millennium systems used 3-digit "curly brace" codes for special characters and diacritics (e.g. "**{226}**" for a non-spacing acute). These codes *were* stored in the field, and the braces had to be escaped when searching them with a regex. This may be what for led to the misstatement in the Sierra manual.

## How Regular Expressions "See" MARC Fields

An unfortunate characteristic of searching MARC data in Create Lists is that the field being searched—what the regex engine "sees"—can vary somewhat depending on how the field is defined in the search statement. It can also vary when you search with the case-sensitive "matches (exact)." This is particularly a problem when you use the start of field position indicator ("**^**").

With MARC variable-length fields, the field to be searched may be defined with a field group tag ("t"=TITLE, "p"=IMPRINT, "d"=SUBJECT, etc.), or it may be defined by MARC tag (type "!" and enter the tag number). In either case, the MARC tag number is usually included at the beginning of what is being searched, that is, the start of field position ("^") is followed by the 3-digit MARC tag. When searching by MARC tag, you may also specify subfields (e.g., "300|bc"), in which case the field to be matched against includes only those subfields.

I tend to search by field group tag and, if necessary, specify the MARC tag, etc. in the regex itself.


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

The following bibliographic MARC field has field group tag “p” (IMPRINT), MARC tag 264, 1st indicator blank, 2nd indicator “1”, and “|a” as the first subfield code:

```
p  264  1|aSan Marino, CA :|bHuntington Library,|c1953.
```

Using this MARC field as an example, the following shows how it would appear to the regex engine under different fields definitions and conditions (as tested in Sierra 5.2). It should be noted that the information shown here has changed through various releases of Sierra, and may continue to do so.

| **Field** | **Condition** |
|---|---|

IMPRINT [p] matches

```
264 1|asan marino,ca :|bhuntington library,|c1953.
^                                                $
```

MARC Tag 264 matches

```
2641san marino,ca :|bhuntington library,|c1953.
^                                              $
```

MARC Tag 264|ab matches

```
san marino,ca :|bhuntington library,
^                                   $
```

IMPRINT [p] matches (exact)

```
San Marino,CA :|bHuntington Library,|c1953.
^                                          $
```

MARC Tag 264 matches (exact)

```
2641San Marino,CA :|bHuntington Library,|c1953.
^                                              $
```

MARC Tag 264|ab matches (exact)

```
San Marino,CA :|bHuntington Library,
^                                   $
```

The variations shown above can be a problem when your regex needs to take into account the beginning of the field, that is, the MARC tag, indicators, and initial subfield code. Note that when searching by MARC tag, any blank indicators are collapsed, along with the initial subfield delimiter (but not subsequent delimiters), causing the remainder of the field to shift left. When specifying particular subfields, the initial subfield delimiter is again excluded.

Note also that when using the field group tag with a “matches (exact)” search, the MARC tag, indicators, and initial subfield delimiter are completely absent. If you need to do a case-sensitive search and also account for the MARC tag, define the field to be searched using the MARC tag.

In the MARC format, the 00X fields do not use indicators or subfield codes. Here too how you search makes a difference. Using a control number (OCLC #) in MARC tag 001 as an example:

| **Field** | **Condition** |
|---|---|

OCLC # [o] matches

```
001  ocn960056509
^            $
```

MARC Tag 001 matches [and matches (exact)]

```
001ocn960056509
^             $
```

OCLC # [o] matches (exact)

```
ocn960056509
^           $
```

*Note: In Sierra, the 006, 007, and 008 fields cannot be searched as such, even with a regex. Instead, you must select the specific component field you want, for example 008 - Date One, or 006 - TypeCode.*


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

## Examples of Regular Expressions in Create Lists

*(Unless otherwise specified in the following examples, the record type is Bibliographic.)*

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 item records where the status is "m" (missing), "n" (billed/not paid), or "$" (lost and paid).

Solution:

**ITEM STATUS matches [mn$]**

---

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.
                               ^
```

Solutions:

**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} and ? quantifiers

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

Solution *(requires ERE)*:

```
     ORDER NOTE  matches  (?e)cancel{1,2}ed
          TITLE  matches  (?e)Thomas (Alva ){0,1}Edison
```

Or:

```
          TITLE  matches  (?e)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 *(requires ERE)*:

```
     MARC Tag 020|c  matches  (?e)\$[1-9]?[0-9]\.[0-9]{2}
```

This matches the dollar sign, followed by a number (1-9), then a second number (0-9), then a decimal point and 2 more numbers. 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 (“\”).


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 *(2nd statement requires ERE)*:

```
          SUBJECT  matches  ^6...[^057]
     OR   SUBJECT  matches  (?e)^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.

---

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 *(requires ERE)*:

```
          MARC  matches  (?e)^035..\|a[0-9]{4}$
     OR   MARC  matches  (?e)^035..\|a[0-9]{4}[^0-9]
```

This search matches |a (note that the delimiter must be escaped with ERE), 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 the 035 might be different in your library.)

---

Example 10

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 has been greatly simplied with the addition of the “AND NOT” operator in Sierra release 5.2:

```
          MARC Tag 040  has      |b
AND NOT   MARC Tag 040  matches  |beng
```

This is possible without the “AND NOT” operator, but it 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 11: 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**" (without a trailing space) in Sierra. Therefore, the following search, intended to match places in the United States:

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

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"]

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

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

---

Example 12

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

```
245 04|aGrand Tour :|bthe lure of Italy in the eighteenth century
245 03|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 non-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 13

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

Solution *(requires ERE)*:

```
MARC Tag 245|a  matches  (?e).{200}.{100}
```

The maximum value for any quantifier is 255 in Sierra (127 in Millennium), 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 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 *(requires ERE)*:

```
MARC  matches  (?e)^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 one or more characters that are not "|". The "+" after the right parenthesis 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." However, 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 fine on the single-character MAT TYPE field.

---

Example 17: Using alternation

Problem: Find titles that are translations from Latin, Greek, or ancient Greek. (MARC tag 041|h is for the code of the original language.

Solution *(requires ERE)*:

**MARC Tag 041|h matches (?e)lat|gre|grc**

Another version:

**MARC Tag 041 matches (?e)\|h(lat|gre|grc)**

---

Example 17: Using alternation

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

Solution *(requires ERE)*:

**300|a matches (?e)^(.*[^0-9])?([1-9]|1[0-9]|2[0-5]) (p\.|page)**

Grouping (parentheses) is used to separate 2 sets of alternative strings. The first alternation gives 3 alternatives: a single-digit number 1-9, *or* a 2-digit number 10-19 *or* a 2-digit number 20-25. Following a space, the second grouping allows either “p.” or “page” to satisfy the match. Note that “page” does not preclude there being more characters following, i.e. “pages”.

Including “**^(.*[^0-9])?**” at the beginning allows preliminary characters to occur optionally, as long as the last of these characters is not a number. This allows matches on values such as “vi, 23 p.” while ensuring that values such as “623 p.” are not matched. Unfortunately, this search will also match, for example, “123 p., 16 p. of plates”, and will fail to match “[8] pages”.

---

Example 18

Problem: Find bib records where the title statement uses ALL CAPS.

Solution *(requires ERE and the “matches (exact)” condition)*:

**MARC Tag 245 matches (exact) (?e)^[^a-z]+(\|[bc][^a-z]+)*$**

The first part, “**^[^a-z]+**”, matches 1 or more characters that are not lower-case letters. The grouping that follows it may occur any number of times, including not at all. It covers any additional subfields, which comprise a delimiter, subfield code “b” or “c”, and 1 or more characters that are not lower-case letters. This is carried to the end of the field (“**$**”). This will match:

```
245 10 |aPHOTOGRAPHY'S RESPONSE TO CONSTRUCTIVISM.
245 10 |aSOUTHERN CALIFORNIA :|bTHE LAND OF FRUIT AND FLOWERS.
```

As described in an earlier section, when using “matches (exact),” the MARC tag, indicators, and initial subfield code are not included in the data being searched. Therefore, the initial subfield (“|a”) with its lower-case “a” does not prevent the match.


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

Example 19: Using back references

Problem: Find repeated words in notes (such as a summary note/abstract)

Solution *(must be done with BRE; back references don't work with ERE)*:

**MARC Tag 520 matches \([a-z]\{5,10\} \)\1**

It's relatively easy to find a particular word that's repeated (e.g., "**NOTE has 'the the '**"). To find *any* word repeated, a back reference must be used to "capture" a word and write that word back into the expression. The grouping in parentheses captures a word. (Because BRE must be used here, the parentheses must be escaped: "**\(…\)**".) A word here contains only letters followed by a space, and I've arbitrarily limited the length of the word to 5-10 characters to reduce the number of hits. (The braces in the quantifier must also be escaped: "**\{5,10\}**".) The expression "**\1**" writes the captured word back in, causing a match to occur if the captured word is repeated:

```
520 … became a luxury item that commanded commanded loyalty …
520 … regarding the ships which which the Golden Fleece did business …
520 … illustrations including including sketches of such luminaries …
520 … who was a great great great granddaughter of William Clift.
```

Quite often, repeated words are not errors, as shown in the last example above.

This search included words made up of only ASCII letters. To broaden the search to include most diacritics, you can add the Latin-1 Supplement and Latin Extended-A characters to the character class using a range expressed with Unicode notation: "**[a-z{u0080}-{u017f}]**".

---

Example 20: Using the AND NOT operator

Problem: Find invalid patron email addresses

Solution: With all the possible ways an email address might be invalid, it would be very difficult to create a regex that would find them all. Fortunately, we can now write one to match a valid address and use the "AND NOT" Boolean operator to retrieve the ones that don't match. Writing such a regex is itself a challenge. Here is one solution:

**PATRON Email Address not equal to** <nothing>
**AND NOT PATRON Email Address matches** ↓

**(?e)^[0-9a-z]([-.0-9a-z_]*[0-9a-z_+])*@([0-9a-z][-0-9a-z_]*\.)+[a-z]{2,4}$**

The above regex was modified from one given in *Regular expression pocket reference*, 2nd edition, by Tony Stubblebine (Sebastopol, CA: O'Reilly, 2007). It's a slight improvement over one I initially came up with for the presentation, although the results with both queries were the same:

```
No access to email      asmith@Caltech
jgomez@wellesey@edu     vjohnson@.ucla.edu
```

(The above examples were modified to disguise the names.)


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

Example 21: Using the AND NOT operator

Problem: Find errors in capitalization and syntax in Library of Congress call numbers

Solution: Like the previous example, this query uses "AND NOT" to find call numbers that do not match the regex. The "matches (exact)" condition is used to find errors in letters case.

```
          CALL #  matches          ^0[59]0
AND NOT   CALL #  matches (exact)   ↓
  (?e)^[A-HJ-NP-VZ][A-Z]{0,2}[1-9][0-9]{0,3}(\.[0-9]+)?
  (\|b\.[A-Z][0-9]+|\.[A-Z][0-9]+\|b[A-Z][0-9]+)
```

This particular query looks at call numbers in either MARC tag 050 or 090. The second line of the regex is an alternation that matches either the 1-cutter option or the 2-cutter option. Nothing is accounted for after the last cutter, although it would be possible to add expressions for an optional year, optional work letters, and optional volume numbers.

It should be noted that there are several areas of the LC Classification Schedule where this pattern is not followed (e.g., maps, regimental histories, some government documents, etc.), and these call numbers will turn up in the results even though they are valid. However, it is not too difficult to pick out errors such as these:

| | |
|---|---|
| `BX8608\|b.c37` | [lower case "c"] |
| `E185.97\|b. P52 1923` | [extra space character] |
| `DA 950\|b.E39` | [extra space character] |
| `N5055\|bC6 1974b` | [missing period] |
| `E195\|b..K28` | [2 periods] |

## More information

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. This book, while probably containing more than you need, is considered the definitive work on the subject:

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

Other useful books from O'Reilly:

Fitzgerald, Michael. *Introducing Regular Expressions*, 1st edition. Sebastopol, CA : O'Reilly, 2012. 154 pp.

Stubblebine, Tony. *Introducing Regular Expressions: Unraveling Regular Expressions, Step-by-Step*, 1st edition. Sebastopol, CA: O'Reilly, 2007. 128 pp.

Here is a website where you can try out regular expressions: https://www.regexpal.com/


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