(Still) Playing with Matches

An update to using regular expressions in Create Lists

Richard V. Jackson
The Huntington Library,
Art Museum, and Botanical Gardens

Thursday, March 25, 2021

Purpose of this presentation

- Introduction to using regular expressions in Create Lists
- Cover the changes in recent releases of Sierra
  - Basic regular expressions (BRE)
  - Extended regular expressions (ERE)
- Provide many useful examples
- Discuss other recent changes in Create Lists
  - Case-sensitive queries
  - "AND NOT" Boolean operator

What are regular expressions (regexes)?

- Specialized text processing tool for matching patterns of characters
  - ➢ Find barcodes that are the wrong length or contain invalid characters
  - ➢ Find invalid email addresses
  - ➢ Find invalid subfield delimiters, wrong indicators, and other MARC coding errors
  - ➢ Simplify searches for variant spellings or alternate character strings
- Widely used in many programming languages and applications
- Use certain special characters (metacharacters) to perform various functions within the regex
- More akin to the "has" condition than the "equal to" condition – record is retrieved if the field being searched *has* (contains) characters matching the pattern of the regex
- Can be used with any field—MARC and non-MARC, variable-length and fixed-length (except date fields) — in any type of record

Some history

- Have been part of Create Lists since the character-based Innopac
- Prior to Sierra release 4.1 were based on the UNIX “egrep” standard
  (egrep = *extended* global regular expression print)
- Beginning with Sierra 4.1 based on the POSIX standard — more compatible with PostGreSQL
- POSIX standard comes in different flavors:
  - Basic Regular Expressions (BRE) — similar to ‘grep’
  - Extended Regular Expressions (ERE) — similar to ‘egrep’
- Default in Create Lists is the less functional BRE
- Possible to use ERE — but the Sierra manual does not mention this

Basic Regular Expressions

Character classes – the period: .

- Period (or “dot”) matches any single character

BIBLIOGRAPHIC	COUNTRY	matches	..c

matches titles published in Canada

ITEM	LOCATION	matches	.vid.

matches item location codes such as *bvid*, *rvid*, *hvidb*

BIBLIOGRAPHIC	SUBJECT	matches	|.maps

matches any subject heading with a subfield delimiter followed by any subfield code followed by “maps” (upper or lower case)

Character classes: […]

- Used to represent any one character that is a member of a user-defined set:

`[abcnp]`	matches any one of the letters *a*, *b*, *c*, *n*, or *p*
`['"]`	matches a single quote or double quote
`[a-z]`	matches any letter (upper or lower case)
`[A-Za-z]`	with case-sensitive searching, matches upper and lower-case letters
`[a-z0-9]`	matches any letter or number
`[14-79]`	matches any of the numbers 1, 4, 5, 6, 7, or 9
`[- ,.]`	matches a hyphen, space, comma, or a literal period *[Note: to be treated as a literal character, the hyphen must be listed first or last]*

Character classes: […]

ITEM	STATUS	matches	[mn$]

matches items with status *missing*, *billed/not paid*, or *lost and paid*

BIBLIOGRAPHIC	CALL #	matches	196[7-9]

matches call numbers containing 1967, 1968, or 1969, for example:

```
F574.D4|bF2 1969
PN1968.G7|bD68 2012
QK15|b.G744 1967
```

Negated character classes: `[^...]`

- Used to represent any character that is *NOT* one of the listed characters

[^]	matches any character that is not a space
[^0-9]	matches any character that is not a number
[^avx-z]	matches any character except *a*, *v*, *x*, *y*, or *z*

	BIBLIOGRAPHIC	BCODE3	matches	[^lnz]	
AND	BIBLIOGRAPHIC	CAT DATE	between	02-01-2021	02-28-2021

At my library, BCODE3 codes *l*, *n*, and *z* are suppression codes. This search retrieves records cataloged in February 2021 that are *not* suppressed.

Negated character classes: [^...]

- Negated character classes are very useful for finding errors.

For example, in MARC tag 650 the valid subfield codes for LCSH are
a, *v*, *x*, *y*, *z* (and possibly *0*-*3*, *6*, *8*)

BIBLIOGRAPHIC	MARC TAG 650	matches	|[^avxyz]

This will match fields such as:

```
650  0 |aSpecial libraries|California.

650  0 |aSculpture, English|d18th century|vExhibitions.
```

Quantifiers – the asterisk (“star”): *

- Quantifiers do not themselves represent characters
- They are given following a character or character class, allow that character or character class to occur some number of times
- The asterisk (*) indicates the preceding may occur any number of times, that is, 0+ times

BIBLIOGRAPHIC	MARC TAG 099	matches	dvd *0*42[0-9][0-9]

matches fields such as:

```
099   |aDVD 004278
099   |aDVD  4225
099   |aDVD04210 pt.2
```

- The asterisk is the only quantifier available under Basic Regular Expressions

A useful combination – dot-star: .*

- "Dot-star" = any number of any characters
- Use as "filler" between more specific expressions

	BIBLIOGRAPHIC	MARC TAG 245	matches	|b.*|b
OR	BIBLIOGRAPHIC	MARC TAG 245	matches	|c.*|c

matches fields such as:

```
245 10 |a1876 :|ba novel /|by Gore Vidal.

245 10 |aCalifornia :|ca history /|cAndrew F. Rolle.
```

Position indicators – end of field: $

- The “$” position indicator “anchors” whatever precedes it to the end of the field
- It should always be given last in the regular expression

BIBLIOGRAPHIC	CALL #	matches	196[7-9]$

matches: `F574.D4|bF2 1969`
does *not* match: `PN1968.G7|bD68 2012`

BIBLIOGRAPHIC	MARC TAG 245	matches	[^.?!]$

matches title fields (245) where the last character is NOT “.”, “?”, or “!”

BIBLIOGRAPHIC	MARC TAG 100	matches	|d18[0-9][0-9]-$

matches personal name authors (100) with birthdates in the 1800s followed by a hyphen and the end of the field (i.e., no death date), e.g.:

```
100 1  |aMcIntyre, Benjamin Franklin,|d1827-
```

Position indicators – start of field: ^

- The “^” position indicator “anchors” whatever follows it to the beginning of the field
- It should always be given first in the regular expression

`PATRON`	`BARCODE`	`matches`	`^[^2]`

matches patron barcodes where the first character is not a ‘2’

Note that the “**^**” metacharacter does double duty:

^ = start of field

`[^...]` indicates a negated character class

`BIBLIOGRAPHIC`	`IMPRINT`	`matches`	`^26[04][23]`

With variable-length fields in MARC format, the field begins with the MARC tag. This search finds imprint fields in tag 260 or 264 with 1st indicator 2 or 3.

“Escaping” a metacharacter: \…

- When you want to treat a metacharacter as a literal character, you must “escape” it, that is, precede it with a backslash (“\”)

BIBLIOGRAPHIC	NOTE	matches	\.\.\.$

matches Note fields that end with 3 periods

	AUTHORITY	NAME AUTHR	matches	^100.***
OR	AUTHORITY	NAME S FRM	matches	^400.***

matches name authority records in which the preferred name (100) or a see reference (400) contains two (or more) literal asterisks:

```
100 0  Monsieur de ******
400 1  H*****x,|cCountess of
```

Extended Regular Expressions

Extended Regular Expressions (ERE)

- To switch to an ERE, precede the expression with: `(?e)`
- Using ERE enables additional metacharacters and metasequences

Metacharacters used in BRE (they still work in ERE):

.	Any character
[…**]**, **[^**…**]**	User-defined character classes
*****	Preceding can occur any number of times
^ … **$**	Start of field and end of field positions
\	Treat the following character as a literal

Additional metacharacters used in ERE:

+	Preceding occurs one or more times
?	Preceding is optional (occurs 0-1 times)
{*min,max***}**, **{***num***}**	Preceding occurs set number of times
(… **)**	Grouping
… **|** …	Alternative strings

Quantifiers: + ?

+	Preceding character or character class occurs one or more times
?	Preceding character or character class is optional (occurs 0-1 times)

BIBLIOGRAPHIC	MARC TAG 035	matches	(?e)[0-9]+ repro

matches 035 tags containing 1 or more numbers followed by <space>"repro"

BIBLIOGRAPHIC	DESCRIPT.	matches	(?e)colou?r

matches Description fields (field tag 'r') containing either "color" or "colour"

IUG 2021 DETROIT

Quantifiers: {*num*} {*min*, *max*}

{*min,max*}	Preceding character/character class occurs at least *min* and no more than *max* times
{*num*}	Preceding character/character class occurs exactly *num* of times

PATRON	BARCODE	matches	(?e)^.{1,13}$

Patron barcode contains 1-13 characters between the start and end of field

PATRON	BARCODE	matches	(?e).{15}

Patron barcode contains 15 characters (it could contain more).

Grouping: (…)

- Use parentheses to group a string of characters to be treated as a unit
- Grouping can be used to apply a quantifier to a string of characters:

```
(?e)hillary (rodham )?clinton
```

matches "Hillary Clinton" or "Hillary Rodham Clinton"

And as if things weren't complicated enough already …

It's possible to use grouping and "{…}" quantifiers under BRE — but you have to escape the parentheses or braces: `\(…\)` `\{`*min*`,`*max*`\}` `\{`*num*`\}`

Under BRE, the above search would be:

```
hillary \(rodham \)\{0,1\}clinton
```

("?" is a literal under BRE, so you have to use "\{0,1\}" instead.)

More on this ahead when I discuss back references …

Alternation: | (vertical bar)

- The vertical bar (“|”) functions as a kind of OR—allowing any of 2 or more alternative strings to satisfy the match. (Remember, with ERE you must escape the subfield delimiter — “\|”)

BIBLIOGRAPHIC	LANG	matches	(?e)fre|ita|spa

matches titles in French, Italian, or Spanish

- Use grouping to separate the alternative strings from the rest of the expression

“this and|or that”	=	“this and” OR “or that”
“this (and|or) that”	=	“this and that” OR “this or that”

BIBLIOGRAPHIC	NOTE	matches	(?e)includes bibliograph(y|ies|ical)

Alternation: examples

- Finding bad non-filing indicators in the 245 tag:

	BIB	TITLE	matches	(?e)^245.0\\|a"?(a |an |the)
AND	BIB	LANG	equal to	eng

MARC tag 245 with 2nd ind 0

Subfield "|a"

Optional double quote

"a ", "an ", or "the "

- Alternative strings can include metacharacters and complex expressions:

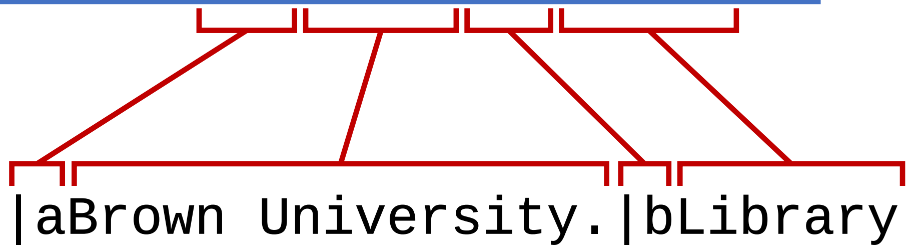
	BIB	TITLE	matches	(?e)^245.0\\|a"?(l'|l[ae] |les |une?)
AND	BIB	LANG	equal to	fre

"l'" "la ", "le ", "les ", "un ", or "une "

A useful metasequence: `[^|]+`

- With MARC variable-length fields, use "`[^|]+`" (or "[^|]*") to match the *content* of any subfield
- This search matches corporate body authority records that have exactly one subordinate body:

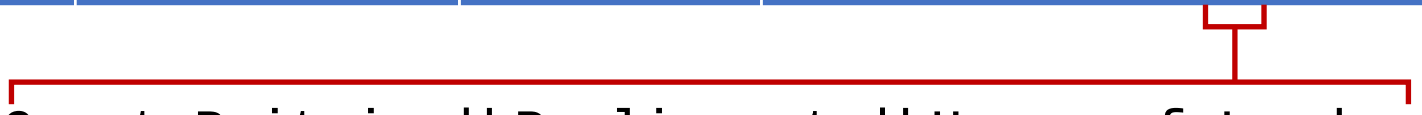
AUTHORITY	NAME AUTHR	matches	`(?e)^110..|a[^|]+|b[^|]+$`



```
110 2  |aBrown University.|bLibrary
```

- On the other hand, this search will also match headings with 2 or more subfield b's:

AUTHORITY	NAME AUTHR	matches	`(?e)^110..|a.+|b.+$`



```
110 1  |aGreat Britain.|bParliament.|bHouse of Lords.|bRecord Office
```

- The expression "`.+`" will match as much as it can without causing the match to fail

A practical example

- Looking at the problem of incorrect non-filing indicators from a different perspective:

```
245 14 |aThe art of D.H. Lawrence /|cby Keith Sagar.
245 14 |aEin Leben den Büchern gewidmet.
245 14 |aLe "Comus" de John Milton, masque neptunien.
245 04 |aPoblacht na h-Eireann =|bThe republic of Ireland.
```

- The above 245 tags all have non-filing indicator 4. The first 3 titles have a space or a double quote as the 4th character following “|a”; the last title has an incorrect non-filing indicator.

BIBLIOGRAPHIC	TITLE	matches	^245.4|a...[^ "']

matches 245 tags with 2nd indicator 4 in which the 4th character of the title is not a space, double quote, or single quote/apostrophe, e.g.:

```
245 04 |aPoblacht na h-Eireann =|bThe republic of Ireland.
```

- You can construct similar queries for non-filing indicator 2, 3, 5, etc.

Back references

- Back references are used to “capture” and store a string of characters and then refer back to it
- Typically used in find/replace operations, which Create Lists cannot do
- Back references can now be used in Create Lists — but only under BRE!
- Capture a string using grouping: `\(…\)`
- Refer back to it with `\1` (or \2, \3, … \9 for subsequent groupings)

BIBLIOGRAPHIC	MARC Tag 520	matches	`\([a-z]\{4,\} \)\1`

Grouping captures a word of 4+ characters plus a space

The back reference enables the match if the same string follows

520 Cuban tobacco and cigars became a luxury item that **commanded commanded** loyalty …
520 … and many illustrations **including including** sketches of such luminaries as …
520 … who was a **great great great** granddaughter of William Clift.

Back references

- Earlier I showed a regex to find a particular subfield code repeated (e.g., "… matches |b.*|b")
- Back references can be used to find multiple occurrences of *any* subfield code

BIBLIOGRAPHIC	AUTHOR	matches	^100.*\(|.\).*\1

matches:
```
100 0   Edward,|cVI|cKing of England,|d1537-1553.
100 1   Saint-Germain,|ccomte de,|d-1784|c(Spirit)
```
[this one's legit]

- Knowledge of MARC is needed — many subfields *are* repeatable
- This query on the 245 looks for subfield codes other than *n* and *p* (which are repeatable):

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

matches:
```
245 10 |aEpigrammatum Graecorum /|annotationibus Ioannis …
245 10 |aElizabeth Harbert papers,|f1848-1950,|f(bulk 1880-1925).
```

Using Unicode notation

- In Sierra, special characters, symbols, and characters with diacritics may be entered using a special Unicode notation – **{u####}** – where “####” is the hexadecimal code for the character:

`{u00a9}` = ©　　`{u014d}` = ō　　`{u00a3}` = £　　`{u0152}` = Œ

- This Unicode notation may be used in a regular expression.
  - The Sierra manual states that you must “escape” the braces – e.g. \{u00A8\}. ***This is wrong!***
  - Characters represented in Unicode notation are interpreted and converted to the actual character prior to processing by the regex engine

BIBLIOGRAPHIC	**MARC Tag 300**	**has**	**{u2113}**

matches: `300 |a 2 p.ℓ., iii, [1], 21, [1] p., 1 ℓ. ;|c23 cm`

BIBLIOGRAPHIC	**MARC Tag ???**	**matches**	**[{u201c}{u201d}]**

matches *any* MARC field (tag ???) containing “curly” double quotes – **“** or **”**

Using Unicode notation

- Other invalid characters and dirty data I occasionally check for:

{u01C2} = ǂ	(OCLC subfield delimiter)
{u2018}, {u2019} = ‘ , ’	(left and right single quote/apostrophe)
{u2013} = –	(en dash)
{u2014} = —	(em dash)
{u00A0} =	(non-breaking space [HTML &nbsp;])

- You cannot use regex metacharacters within Unicode notation. *Expressions like this don’t work*:

```
{u03..}      {u03[7-9a-f][0-9a-f]}
```

- However, you may use a range of specific Unicode values in a character class:

BIBLIOGRAPHIC	MARC Tag 880	matches	[{u0370}-{u03ff}]

matches 880 tags containing characters from the Greek/Coptic character set

- Handy website for looking up Unicode values: https://unicode-table.com/

Case-sensitive queries in Create Lists

- Introduced in Sierra 5.0. Four new conditions now distinguish upper and lower-case letters:
  - has (exact) [*keyboard input:* x]
  - matches (exact) [j]
  - starts with (exact) [s]
  - ends with (exact) [z]

BIBLIOGRAPHIC	MARC Tag 6???0	matches (exact)	|[vxyz][a-z]

matches 6xx tags with 2nd ind. 0 where a subdivision begins with a lower-case letter:

```
650  0 |aViaducts|zFrance|xhistory|y19th century
651  0 |aPanama|xMaps|yca. 1914?
600 10 |aToler, Henry,|d-1824.|tUnion|xno union.
                        [used to be: “Union--no union”]
```

Odd bug: Case-sensitive searching using any of the “exact” conditions doesn’t work with c-tagged fields in Bib or Item records. It does work if you search by MARC tag (e.g. 050)

IUG 2021 DETROIT

How regular expressions “see” variable-length MARC fields

- A complication in searching MARC fields is that they appear differently depending on whether you’re searching by field group tag or MARC tag, or using “matches” or “matches (exact)”

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

BIBLIOGRAPHIC IMPRINT matches … sees the above field as:

```
2 6 4   1 | a s a n   m a r i n o ,   c a   : | b h u n t i n g t o n   l i b r a r y …
```

BIBLIOGRAPHIC MARC Tag 264 matches … see the same field as:

```
2 6 4 1 s a n   m a r i n o ,   c a   : | b h u n t i n g t o n   l i b r a r y …
```

BIBLIOGRAPHIC IMPRINT matches (exact) … sees the field as:

```
S a n   M a r i n o ,   C A   : | b H u n t i n g t o n   L i b r a r y …
```

BIBLIOGRAPHIC MARC Tag 264 matches (exact) … sees the field as:

```
2 6 4 1 S a n   M a r i n o ,   C A   : | b H u n t i n g t o n   L i b r a r y …
```

Case-sensitive queries: finding titles in ALL CAPS

- Problem: Find titles that were keyed in using ALL CAPS.
- Solution: Use the “matches (exact)” condition, but rather than focus on “**`[A-Z]`**” plus other characters that might be present, search for “**`[^a-z]`**” – the absence of lower-case letters

BIBLIOGRAPHIC	TITLE	matches (exact)	(?e)^[^a-z]+$

matches:

```
  245 00 |aGENERAL PATTERSON'S CAMPAIGN IN VIRGINIA.
  245 10 |aPHOTOGRAPHY'S RESPONSE TO CONSTRUCTIVISM.
```

A better version than allows additional subfields (subfield codes are lower-case):

BIB	TITLE	matches (exact)	(?e)^[^a-z]+(|[bcnp][^a-z]+)*$

matches:

```
  245 10 SOUTHERN CALIFORNIA :|bTHE LAND OF FRUIT AND FLOWERS.
  245 10 LIBERTY :|bTHE FRENCH AMERICAN STATUE IN ART AND HISTORY.
```

Using the “AND NOT” Boolean operator

- Introduced in Sierra 5.2 (Idea Lab winner of the Create Lists challenge in 2019)
- Makes it possible to retrieve records that do not match a particular regex:

	PATRON	Barcode	not equal to	
AND NOT	PATRON	Barcode	matches	(?e)^[0-9]{14}$

matches patron barcodes that are longer or shorter than 14 digits (or that contain non-numeric characters within the 14 characters)

- Without Boolean NOT, the above query would require 3 separate statements:

	PATRON	Barcode	matches	(?e).{15}
OR	PATRON	Barcode	matches	(?e)^.{1,13}$
OR	PATRON	Barcode	matches	[^0-9]

Using “AND NOT”: an example

- Boolean NOT becomes especially useful when the pattern you *don’t want to match* is complex
- Example: It’s possible to write a regex that defines a valid email address, but very difficult to write one for every possible invalid one. Boolean NOT solves this problem. Here is my version of one for a valid email address::

```
(?e)^[a-z0-9_-]+(\.[a-z0-9_-]+)*@([a-z0-9_-]+\.)+[a-z]{2,4}$
```

Features of this regex:	Uses start and end of field (^ … $) to account for entire field; only one “@”
	Name and domain components contain only letters, numbers, “_”, “-”, and “.“
	Periods cannot appear at the beginning, end, or next to the “@” or another period
	Field must end in 2-4 letters for the top-level domain

	PATRON	Email	not equal to	
AND NOT	PATRON	Email	matches	*[the above regex]*

- This search will match email addresses such as:

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

Using “AND NOT”: another example

- A similar technique could be used to find syntax errors in Library of Congress call numbers
- Here is a regex (to be used with “matches (exact)”) that will match most LC call numbers:

```
(?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 query finds bib records with a call number in MARC tag 050 or 090, where the call number *does not match* the above pattern:

	BIB	CALL #	matches	^0[59]0
AND NOT	BIB	CALL #	matches (exact)	*[the above regex]*

- Note: There are many valid LC call numbers that do not fit the typical pattern (e.g., regimental histories, maps) and will be retrieved by this query. But this can help you find errors such as:

```
BX8608|b.c37          E185.97|b. P52 1923
DA 950|b.E39          N5055|bC6 1974b
```

A wish list

- ✓ Make ERE the default
- ✓ Fix the documentation!

  (Sierra Guide > Creating Lists (Review Files) > Using Relational Operators [section on “matches“])
- ✓ Searches on fields defined by MARC tag should be consistent with those using field group tags (do not collapse blank indicators or exclude initial subfield)
- ✓ Searches with “matches (exact)” should include MARC tag, indicators, and the initial subfield delimiter

Conclusion

- POSIX standard has made Create Lists regex more complicated but somewhat more powerful
- Effective use of regular expressions in Create Lists requires …
  - ➢ Understanding regex syntax
  - ➢ Familiarity with your data and local practices
  - ➢ Familiarity with the MARC format
- Learn through practice
  - ➢ Create Lists makes an excellent “sandbox” for learning regular expressions
  - ➢ Skills learned in Create Lists can be applied to other regex implementations

THANK YOU

Questions?

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