



## W3: Regex Rules! Using Regular Expressions with MarcEdit

July 13, 2019

PRESENTED BY

**Yael Mandelstam**

Fordham Law Library  
[ymandelstam@law.fordham.edu](mailto:ymandelstam@law.fordham.edu)

**Jean M. Pajerek**

Cornell Law Library  
[jmp8@cornell.edu](mailto:jmp8@cornell.edu)

**Alan Keely**

Wake Forest University  
Professional Center Library  
[keelyda@wfu.edu](mailto:keelyda@wfu.edu)





## Program Outline

### Regular Expressions: Introduction and Basic Concepts

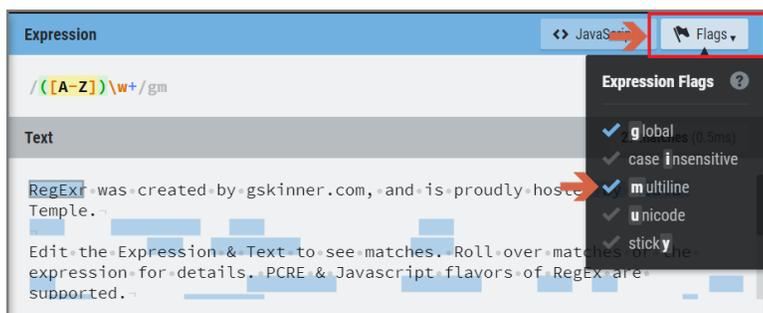
- General Information
  - What is Regex and Why Should We Use It?
  - Regex Resources
  - Online Regex Testing Tools
- Basic Concepts
  - Anchors & Escapes
  - Character Classes
  - Quantifiers & Alternations
  - Grouping (in Find)
  - Substitution (In Replace)

### Brief introduction to MarcEdit

- General Information
- MarcEdit Main Window
- MarcEdit File Formats
- Opening a File of MARC Records
- Regex in the MarcEditor

### Putting It All Together: Using Regular Expressions in MarcEdit

- Best Practices
- Scenario 1: Changing Order of Subfields
- Scenario 2: Adding Missing Subfield Delimiters/Codes
- Scenario 3: Deleting Fields Based on Specific Criteria
- Scenario 4: Adding Missing Punctuations
- Scenario 5: Extracting a Subset of Records Based on Call Number Ranges



In the **Basic Concepts** section we'll be using **RegExr**, a regex online testing tool available at <https://regexr.com/>

In addition to the default **global** flag, select the **multiline** flag. In most cases the global mode, which ensures that the regex tool does not stop after the first match, would suffice. But when using special characters that designate beginning/end of line, the regex tool needs to be in **multiline** mode.

Note that MarcEdit recognizes beginning/end of line characters by default.

# Regular Expressions (regex): Introduction and Basic Concepts

## GENERAL INFORMATION

### What is Regex and Why Should We Use It?

Normally, when we search for a text string, we look for exact matches. For example, when we search for “cat”, we want the first character to be a “c”, the second character to be an “a”, and the third character to be a “t”. But what if we just want the first character to be a capital letter, any capital? We can do 26 separate searches (Aat, Bat, etc.) or we can use a search term to capture a number of different characters.

With regex we are able to look for a search PATTERN when a regular search cannot accomplish what we are trying to do.

Characters in regex can be treated either as regular characters such as **a, b, C, D, 1, 2** and are thus called “literals”, or have special function. Characters with special function in regex are called **METACHARACTERS** or **SPECIAL CHARACTERS**.

## Regex Resources

- Getting started with regex (Terry Reese)
  - Power Point: [https://www.slideshare.net/reese\\_terry/getting-started-with-regular-expressions-in-marcedit](https://www.slideshare.net/reese_terry/getting-started-with-regular-expressions-in-marcedit)
  - YouTube video: <https://www.youtube.com/watch?v=7YXvS4xBEfw&feature=youtu.be>
- RegexOne (interactive regex tutorial): <https://regexone.com>
- Microsoft.net regex site (quick reference and other tools): <https://docs.microsoft.com/en-us/dotnet/standard/base-types/regular-expression-language-quick-reference>

## Online Regex Testing Tools

- **RegExr**: <https://regexr.com/>
- **Regex101**: <https://regex101.com/>
- **RegEx Pal**: <https://www.regexpal.com/>

## BASIC CONCEPTS

### Anchors & Escapes

- ^ (caret)** Matches beginning-of-line position
- \$ (dollar sign)** Matches end-of-line position
- \ (backslash)**
  1. Has a special function when coupled with a specific letter (e.g., `\n` matches new line)
  2. Specifies that a metacharacter should be treated as a literal. In regex this is called “escaping” a metacharacter

#### ^ (caret)

Expression
<code>/^app/gm</code>
Text
apple↵ popsicles↵ grapple↵ lesson↵ application↵

#### Example #1

**^app** matches only the “app” in apple and application

**IMPORTANT:** In some contexts, the caret has a different function. See section on Character Classes.

#### \$ (dollar sign)

Expression
<code>/le\$/gm</code>
Text
apple↵ popsicles↵ grapple↵ lesson↵ application↵

#### Example #2

**le\$** matches the two last letters in apple and grapple

### Useful Tips

The dollar sign (\$) is useful for finding and adding character(s) or strings at end of fields and subfields, e.g., missing punctuation or additional subfields or text.

The dollar sign combined with the caret (^) is useful for establishing the boundaries of the string, e.g., **^abc\$** matches the entire “abc” string and nothing else.

### Exercise 1

apple  
popsicles  
grapple  
lesson  
application

Write an expression that matches only the words that end with **ple**

### \ (backslash)

When coupled with a specific letter the backslash has a special function (e.g., `\d` matches any digit), but when preceding a metacharacter, it does the opposite and “escapes” the metacharacter so it can be treated as a literal (e.g., since the question mark is a metacharacter, we need to escape it with a backslash – `\?` – if we want to use it as a literal)

Metacharacters such as `^ $ \ | . * + ? { } [ ] ( )` need to be **ESCAPED with a backslash** (`\`) to match literal characters

### Exercise 2

How would you express **\$z** in the following field?  
How would you express **\$u**?

856 40\$zOnline Resource\$uhttp://www.heinonline.org

## Character Classes

<b>[abc]</b>	Matches any one of the enclosed characters. <u>Case sensitive</u> , but order does not matter
<b>[^abc]</b>	Matches any character NOT enclosed in the square brackets. In this example every character is matched EXCEPT for “a”, “b”, or “c”. Also case sensitive, but again, order does not matter
<b>[ - ]</b>	A range of characters, e.g., [a-z], [A-Z], [0-9]
<b>. (period)</b>	Matches any single character except line break
<b>\w</b>	Matches any alphanumeric character as well as underscore; same as <b>[a-zA-Z0-9_]</b>
<b>\W</b>	Matches any character that is not alphanumeric or underscore; same as <b>[^a-zA-Z0-9_]</b>
<b>\s</b>	Matches whitespace characters such as space, tab, and new line
<b>\S</b>	Matches non-whitespace characters
<b>\d</b>	Matches any digit; same as <b>[0-9]</b>
<b>\D</b>	Matches any non-digit; same as <b>[^0-9]</b>

Only **^ - \ [ ]** need to be escaped inside a character class to match literal characters

Expression
<code>/gr[ae]/gm</code>
Text
<pre> grey gray grate gravy greet groovy gravity </pre>

### Example #3

This expression looks for matches on the literal letters “gr” followed by EITHER “a” or “e”

Expression
<code>/gr[^ae]/gm</code>
Text
<pre> grey gray grate gravy greet groovy gravity </pre>

### Example #4

Looks for matches on the literal letters “gr” followed by any letter that is NOT “a” or “e”. The caret (^) negates both the “a” and the “e” in the expression.

**Expression**

```
/gr[aeiou]/gm
```

**Text**

```
grey↵
gray↵
grate↵
gravy↵
greet↵
groovy↵
gravity
```

**Example #5**

Looks for matches on the literal letters “gr” followed by **ONE** of the vowels. Notice that the second “e” in “greet” and the second “o” in “groovy” are NOT matched.

**Expression**

```
/gr[a-u]/gm
```

**Text**

```
grey↵
gray↵
grate↵
gravy↵
greet↵
groovy↵
gravity
```

**Example #6**

In this example, instead of listing all the possible vowels, we’re using a range of letters between “a” and “u.” The result is similar to #5, though unlike the previous example, it would also match any other character between “a” and “u” following “gr” (e.g., grp)

**Expression**

```
/gr[aeiou]y/gm
```

**Text**

```
grey↵
gray↵
grate↵
gravy↵
greet↵
groovy↵
gravity
```

**Example #7**

What happens when we add the literal letter “y” to the end of this expression? “grey” and “gray” are matched, but “gravy” and “groovy” are not, even though they have the letter “y” at the end.

**QUESTION: Why is this?**

**Expression**

```
/gr[aeiou].y/gm
```

**Text**

```
grey↵
gray↵
grate↵
gravy↵
greet↵
groovy↵
gravity
```

**Example #8**

A period (.) matches any single character except line feed (\n). What happens when we put one in front of the “y?” The period is a placeholder for the single character “v” in “gravy,” so a match is identified.

Expression
<code>/gr[aeiou].y/gm</code>
Text
<pre>grey gray grate gravy greet groovy gravity</pre>

**Example #9**

Adding a second period allows a match on “groovy” because one period takes the place of the second letter “o” and the other period fills in for the letter “v”

**QUESTION: What will happen if we add a third period (in front of the “y”) to the expression?**

Expression
<code>/gr\w./gm</code>
Text
<pre>grey gray grate gravy greet groovy gravity</pre>

**Example #10**

`\w` represents any alphanumeric character, as well as the underscore character. This expression matches the literal letters “gr” followed by any alphanumeric character (`\w`), followed by any single character (`.`)

Expression
<code>/g\w./gm</code>
Text
<pre>gRay gRey gRate </pre>

**Example #11**

`\w` is not case-sensitive. These matches would still be found if the letter “r” were capitalized in the text we’re matching against.

Expression
<code>/\W/gm</code>
Text
<pre>978-2-275-063560</pre>

**Example #12**

`\W` matches any non-alphanumeric character

The same result can be achieved using `[^a-zA-Z0-9_]`

Expression
<code>/\d/gm</code>
Text
<pre>978-2-275-063560</pre>

**Example #13**

`\d` matches any digit. In this example, all the digits are matched, while the hyphens are not.

The same result can be achieved using `[0-9]`

Expression
<code>/\D/gm</code>
Text
<pre>978-2-275-063560</pre>

**Example #14**

`\D` matches characters that are NOT digits

**QUESTION: What is another way to obtain the same result?**

Expression
<code>/\s/gm</code>
Text
978-2-275-063560

**Example #15**

`\s` matches a whitespace character

Expression
<code>/\S/gm</code>
Text
978-2-275-063560

**Example #16**

`\S` matches non-whitespace characters

**Exercise 3**

Expression
<code>/gr[aeiou]/gm</code>
Text
grey- gray- grate- gravy- greet- groovy- gravity

Consider this sample text and regular expression. Can you think of a way to alter the expression to match the word “gravity” and nothing else?

**Exercise 4**

TMX-482  
GAL-931  
LTU-387  
JOYRIDE  
HTP 638  
IMGR8T  
GODOGGO  
WTX-495

4.1 Here is a short list of license plate numbers. Write an expression that matches the properly formatted ones, i.e., 3 letters, hyphen, 3 digits.

TMX-482  
GAL-931  
LTU-387  
JOYRIDE  
HTP 638  
IMGR8T  
GODOGGO  
WTX-495

4.2 Write an expression that matches the license plate number missing its hyphen.

## Quantifiers & Alternations

<b>*</b> (asterisk)	Matches preceding expression zero or more times
<b>+</b> (plus)	Matches preceding expression one or more times
<b>?</b> (question mark)	Matches preceding expression zero or one time
<b>{n}</b>	Matches preceding expression exactly n times
<b>{n,}</b>	Matches preceding expression n or more times
<b>{n,m}</b>	Matches preceding expression between n and m times
<b> </b> (pipe)	Alternation/OR

**\* (asterisk)**      **+** (plus)      **?** (question mark)

Expression
<code>/.<b>*</b>leg.<b>*</b>/gm</code>
Text
<pre> legal trial illegal mistrial delegate trials leg industrial elegy                     </pre>

### Example #17

The first **.\*** matches any character before “leg” zero or more times

The second **.\*** matches any character after “leg” zero or more times

Expression
<code>/.<b>+</b>leg.<b>+</b>/gm</code>
Text
<pre> legal trial illegal mistrial delegate trials leg industrial elegy                     </pre>

### Example #18

The first **.+** matches any character before “leg” one or more times

The second **.+** matches any character after “leg” one or more times

**Question: Why does this expression not match “legal” and “leg”?**

Expression
<code>/^leg.+ /gm</code>
Text
<pre> legal trial illegal mistrial delegate trials leg industrial elegy                     </pre>

**Example #19**

`^` matches the beginning of line

`.+` matches any characters after “leg” one or more times

Expression
<code>/^.?trial.? /gm</code>
Text
<pre> legal trial illegal mistrial delegate trials leg industrial elegy                     </pre>

**Example #20**

`^` matches the beginning of line

`.?` matches any character before and after “trial” zero or one times

**Question: Why does this expression not match “mistrial” and “industrial”?**

**Exercise 5**

law  
laws  
bylaw  
lawyer

- 5.1 Write an expression that matches all four words
- 5.2 Write an expression that matches ONLY **laws** and **lawyer**
- 5.3 Which word(s) does the expression `.?law.+` NOT match?

**{n}**    **{n,}**    **{n,m}**

Expression
<code>/\d{13} /gm</code>
Text
<pre> 9065440860 9789065440860 069105438X 9780691054384                     </pre>

**Example #21**

`\d` matches any digit

`{13}` matches the preceding expression exactly thirteen times (this expression will match any 13-digit ISBN, as long as there are no spaces or hyphens between the digits)

This expression can also be written as `[0-9]{13}`

**Example #22**

`{n,}` matches preceding expression n or more times

`.m{1,}` matches **am**, **umm**, and **hmmm**  
`.m{2,}` matches **umm** and **hmmm** but NOT **am**

`.m{3,}` matches **hmmm** but NOT **am** and **umm**

`.m{4,}` matches none of the three words

Expression
<code>/45{2,3}6/gm</code>
Text
4556
45556
456
455556

**Example #23**

**{2,3}** matches the preceding expression between two and three times, so when following the number “5”, it matches 55 and 555

In addition to the above, this expression also requires the string to have first digit “4” and last digit “6”

**Question: Why does this expression not match “456” and “455556”?**

**Exercise 6**

67805  
678805  
6788805  
67888805

Which of these numbers are NOT matched by the expression **678{1,3}05**?

**Exercise 7**

pizza  
piazza  
puzzle  
pizzeria

7.1 Write an expression that matches **puzzle** and **pizzeria** only  
7.2 Which word(s) does the expression **. \*zz.{1,2}** NOT match?

## | (pipe)

It is best practice to ALWAYS put alternations (OR) in parentheses to ensure the matching only of that part of the expression you want matched by the alternation. For example, think of the difference in search results for “white tables OR chairs” vs. “white (tables OR chairs)”

Expression
<code>/.*(le lo)\$/gm</code>
Text
apple armadillo balloon urban popsicle local banana leaves

**Example #24**

Matches entries that end with “le” OR “lo”

Expression
<code>/^ (app ban).*/gm</code>
Text
apple armadillo balloon urban popsicle local banana leaves

**Example #25**

Matches entries beginning with “app” OR “ban”

**Exercise 8**

```
=035 \\$a(OCoLC)ocm99988765
=035 \\$a(OCoLC)ocn499887655
=035 \\$aocm99988765
=035 \\$aocn499887655
```

Write an expression that matches OCLC numbers that begin with **ocm** OR **ocn** and do not have the **(OCoLC)** prefix

**Exercise 9**

Find at least three ways to express the 246 field up to the first subfield:

**=246[space][space][indicator][indicator]\$**

## Grouping & Substitution

**Grouping (in Find):** In addition to using parentheses to restrict the application of certain functions to a specific part of the expression, they are also used for grouping substrings that can later be referenced in Replace (first group will be referenced as \$1, second group as \$2, etc.). These grouped substrings are often referred to as “subexpressions” or “capturing groups”.

**Substitution (in Replace):** Once a search pattern is specified in Find, a second expression is used in Replace to indicate what substitutions to make within the matched text. This is done by referencing the results of the search pattern and making the desired changes to that pattern.

### Example #26

The screenshot shows a regex tool interface with three main sections: Expression, Text, and Tools. The Expression field contains the regex `/([md]onkeys) love (bananas|carrots)/gm`. The Text field displays four lines of text: `monkeys love bananas`, `monkeys love carrots`, `donkeys love carrots`, and `donkeys love bananas`. The Tools field shows the replacement string `$1 eat $2` and the resulting text: `monkeys eat bananas`, `monkeys eat carrots`, `donkeys eat carrots`, and `donkeys eat bananas`. A "Replace" button is visible on the right side of the Tools section.

**Group 1: ([md]onkeys):** “m” or “d” followed by “onkeys” matches both “monkeys” and “donkeys”. This group is referenced in Replace as **\$1**

**Group 2: (bananas|carrots):** matches bananas OR carrots  
This group is referenced in Replace as **\$2**

Since we want to replace the word “love” with “eat”, we ignore the former and add the latter to the Replace string, with spaces on each side: **\$1 eat \$2**

#### Useful REPLACE Tips

To add a literal MARC subfield in Replace, use **\$\$** (two dollar signs)

To add a literal number in Replace, put the number after the dollar sign in curly brackets. For example, to add “1” after the first referenced group (\$1), express it as **\${1}1\$3**. Without the curly brackets (**\$11\$3**), the system will think you want group eleven followed by group three.

### **\n (new line)**

The `\n` matches a new line, also known as line feed or line break. It is useful in MarcEdit when we want to insert a new field or split an existing field.

**Important:** When writing an expression in MarcEdit that involves multiple lines, you need to check not only “Use Regular Expressions” but also “Use Multiline Evaluation”.

### **Example #27**

The screenshot shows a vertical interface with three main sections:

- Expression:** Contains the regex `/(.*)\s(.*)/gm`. The `(.*)` and `\s` are highlighted in green.
- Text:** Contains the input string `apples•bananas`, where the dot separator is highlighted in blue.
- Tools:** Contains the replacement string `$1\n$2`, where `\n` is highlighted in purple.

Below the tools section, the output is shown as two lines of text: `apples` followed by `bananas` on a new line.

## Brief introduction to MarcEdit

### GENERAL INFORMATION

- **What is MarcEdit?**

MarcEdit is a free universal library metadata tool, developed and owned by Terry Reese

- **System requirements for MarcEdit 7.x**

Windows 7+ (XP no longer supported)

Windows .NET Framework 4.6+

- **Installing MarcEdit**

Downloads available at <http://marcedit.reeset.net/downloads>

To find out if you need the 32-bit or 64-bit download, click the **Start** button on your computer and click **System** (if it's not listed, it should appear when you type 'system' in the search box). In the box that opens with information about the system, you should find your **system type**.

- **Help/Support**

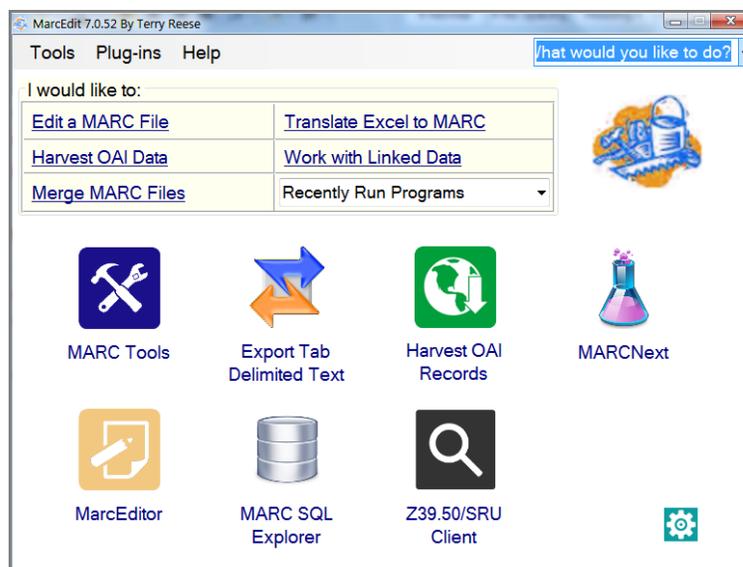
MarcEdit Listserv (subscribe and search list archives): <https://listserv.gmu.edu/cgi-bin/wa?A0=marcedit-l>

Post to MarcEdit Listserv: [MARCEDIT-L@listserv.gmu.edu](mailto:MARCEDIT-L@listserv.gmu.edu)

MarcEdit Tutorials on YouTube: <https://www.youtube.com/user/tpreese/playlists>

Terry Reese [reeset@gmail.com](mailto:reeset@gmail.com)

### MARCEdit MAIN WINDOW



### About MarcEdit File Formats

MarcEdit saves MARC21 files in the following formats:

\***.mrc** for records in raw MARC21 format (binary)

\***.mrk** for records in human-friendly mnemonic format

To convert records from .mrc to .mrk use **MarcBreaker**

To convert records from .mrk to .mrc use **MarcMaker**

MarcEdit can also read other file types such as Innovative Interfaces **.out** files, OCLC **.dat** files, vendor **.001**, **.marc**, and **.bin** files. To use these files in MarcEdit, change the file extension to **.mrc** via **right-click > Rename** or **F2**

### OPENING A FILE OF MARC RECORDS

There are various ways to open a file of .mrc records:

1. Directly from file: Double-click on **filename > Execute > Edit Records**
2. Via MARC Tools  
MarcEdit Main Window > **MARC Tools** icon > Select Operation: **MarcBreaker**  
Select Data to Process: Open...: select your binary (.mrc) file  
Save As...: select location and name mnemonic (.mrk) file  
Click **Execute > Edit Records**
3. Via MarcEditor  
MarcEdit Main Window > **MarcEditor** icon  
Select **File > Open**  
At the bottom of the dialog box select **MARC Files (\*.mrc)** Locate your file and click **Open**

### REGEX IN THE MARCEDITOR

- File > Select Records for Edit
- Edit > Find & Replace
- Tools > Add/Delete Field
- Tools > Copy Field Data
- Tools > Edit Fields
- Tools > Edit Subfield Data
- Tools > Swap Fields
- Tools > Regular Expression Store
- Tools > Build New Field
- Report > Custom Reports

## Putting It All Together: Using Regular Expressions in MarcEdit

*"It is easy to write regular expressions to match what you want and expect. It is much harder to write regular expressions that anticipate all possible scenarios so that they do not match what you do not want to match."*

*Ben Forta*

### BEST PRACTICES

- Always, ALWAYS, ALWAYS keep an unedited copy of your original file
- When testing a new expression, keep track of your various iterations of the expression in a separate text file so you don't end up moving around in circles and repeating the same failed expressions
- Always check each revision by viewing the targeted field BEFORE and AFTER via the Find function:
  - **Edit > Find (Ctrl+F)**
  - Enter field (e.g. =050)
  - Click **Find All**
- Know how to use UNDO:
  - Option 1: Use the Windows-wide **Ctrl+Z** to undo your last edit; use **Ctrl+Y** to reverse your last Undo
  - Option 2: Select **Edit > Undo (F2)**
  - To undo your previous **GLOBAL** edit select **Edit > Special Undo (Ctrl+Alt+F2)**
- When stuck, take advantage of the regex experts on the MarcEdit Listserv. They will not only help you construct your expression, but will often add a detailed explanation that can be very educational.

## SCENARIO 1: CHANGING ORDER OF SUBFIELDS

**Description:** Some of the 856 fields in a file have \$u followed \$z, while others have \$z followed by \$u. We want all fields to follow the same pattern of \$z followed by \$u.

**Solution:** In the MarcEditor select **Edit > Replace (Ctrl+R)**. Check **Use regular expressions**

Find: <b>(=856 ..)(\\$.*)(\$z.*)</b>	<b>Group 1: (=856 ..)</b>	(field 856[space x2][any character x2])	<b>\$1</b> in Replace
Replace: <b>\$1\$3\$2</b> (groups <b>2</b> & <b>3</b> are switched)	<b>Group 2: (\\$.*)</b>	[(dollar sign escaped][u][everything else])	<b>\$2</b> in Replace
	<b>Group 3: (\$z.*)</b>	[(dollar sign escaped][z][everything else])	<b>\$3</b> in Replace

### Exercise 10

Some 040 fields have \$e (Description Convention) appear before \$b (Language of Cataloging). For example:

=040 \\\$aABC\$**erda**\$ **Beng**\$cABC

Write an expression to move \$b before \$e

## SCENARIO 2: ADDING MISSING SUBFIELD DELIMITERS/CODES

**Description:** We have a file of MARC records with several different formats of 041 field. Some have two or three sets of three-letter language codes strung together in one \$a, some have \$b with two language codes strung together, and a few have \$h appended at the end with no data. We want to delete \$h and insert missing delimiters \$a and \$b between language codes.

**Solution:** In the MarcEditor select **Tools > Edit Subfield Data**. Check **Use regular expressions**

**Step 1:** Remove \$h

Field: **041**  
 Subfield: **h**  
 Click **Remove Text**

Note that this step does not require regex

**Step 2:** Add \$a between the first and second language codes in subfield "a"

Field: **041**  
 Subfield: **a**  
 Field Data: **([a-z]{4})([a-z]{3})**  
 Replace with: **\$1\$a\$2**  
 Check **Regular Expression**

**Group 1:** any four lowercase letters (the "a" in \$a + the first three-letter language code); referenced in Replace as **\$1**  
**Group 2:** any three letters (the second three-letter language code); referenced in Replace as **\$2**

**\$a** in Replace inserts the missing \$a between the two language codes. Click **Replace Text**

**Step 3:** Add \$a between the second and third language codes in subfield "a"

Same as step 2, so just click **Replace Text** again.

**Note:** If there are more than 3 language codes in \$a, this step can be repeated multiple times until we get the message "0 modifications were made"

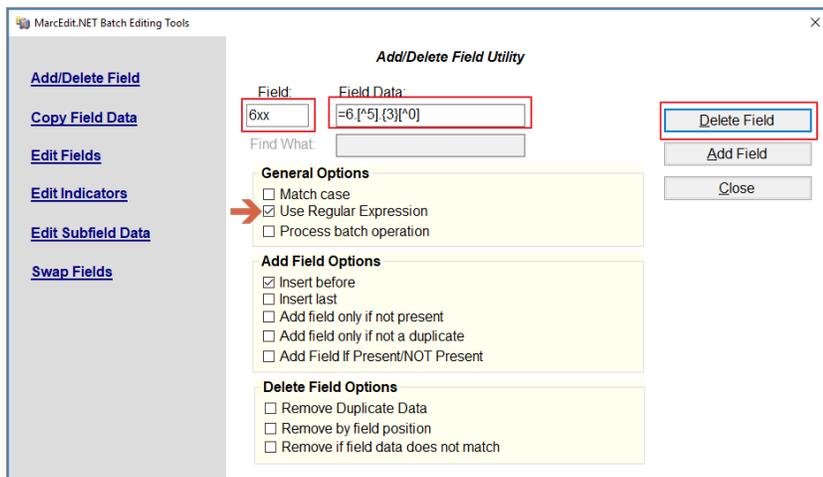
**Exercise 11**

Using **Tools > Edit Subfield Data** Add **\$b** between the first and second language codes in subfield "b"

### SCENARIO 3: DELETING FIELDS BASED ON SPECIFIC CRITERIA

**Description:** We received a file of MARC records that has a mixture of LC and non-LC subject headings. We wish to keep the former (LC 6xx with second indicator “0”) and delete the latter, but want to make sure the 655 fields that do not have second indicator “0” are not deleted.

**Solution:** In the MarcEditor select **Tools > Add/Delete Field (F7)**. Check **Use regular expressions**



- =6** Beginning of 6xx field
- .** (period) Any single character
- [^5]** Any character other than “5”
- {3}** Any three characters
- [^0]** Any character other than “0”

Putting it together: **=6.[^5].{3}[^0]**

Click **Delete Field**

- =650** \4\$aOrganisation internationale.
- =651** \0\$aSalem (Mass.)
- =655** \7\$aStatutes and Codes.\$2lcgft

Only the **first** field fulfils all required criteria and will be deleted. The other two fields will remain untouched.

### Exercise 12

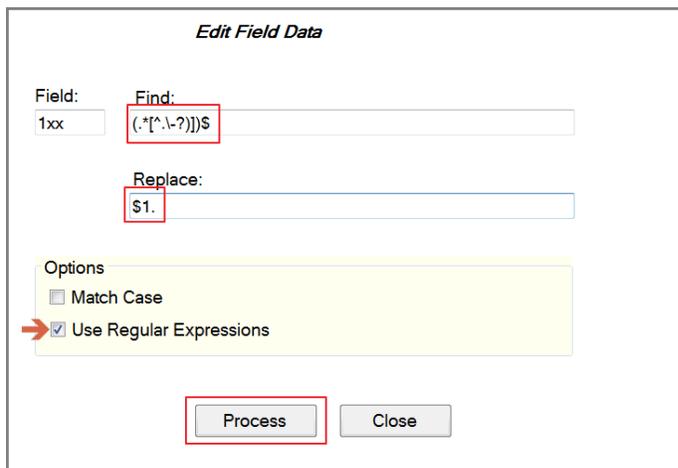
We have a file of MARC records that includes multiple 856 fields with URLs for various domains, including llmc, heinonline, google, hathitrust, ebSCO, and others. We want to keep only 856 with **llmc OR heinonline** and delete all other URLs.

In the MarcEditor select **Tools > Add/Delete Field**

### SCENARIO 4: ADDING MISSING PUNCTUATIONS

**Description:** We received a file of MARC records with some 1xx fields missing the end period. We would like to add that period but ONLY if the field has no end punctuation such as period, hyphen, question mark, or closed parenthesis. Note that the expression below can be used for other fields such as 7xx and 6xx.

**Solution:** In the MarcEditor select **Tools > Edit Field Data (Ctrl+Shift+F3)**. Check **Use regular expressions**



**Find: (.\*[^\.-?])\$**

Remember that elements within square brackets are always interpreted as OR, and that only ^ - \ [ ] need be escaped within the square brackets.

- ( Start group (to be referenced in Replace as \$1)
- .\* Any character(s)
- [^ Open square brackets and add caret. The caret will negate everything within the square bracket:
  - . period (no need to escape),
  - \- or hyphen (escaped),
  - ? or question mark (no need to escape),
  - ) or closing parenthesis (no need to escape).
- ] Close square bracket
- ) Close group
- \$ Match end of field

**Replace: \$1.**

**\$1** Group 1

**.** (period) Added missing period (note that there is no “escaping” in Replace)

Click **Process**

#### Exercise 13

We want to add a slash before 245\$c, but only if it is missing. We tried this expression, but it’s not working. Can you identify the error(s)?

**Find: (=245.\*[^/])(/\$c)**

**Replace: \$1 \ \$2**

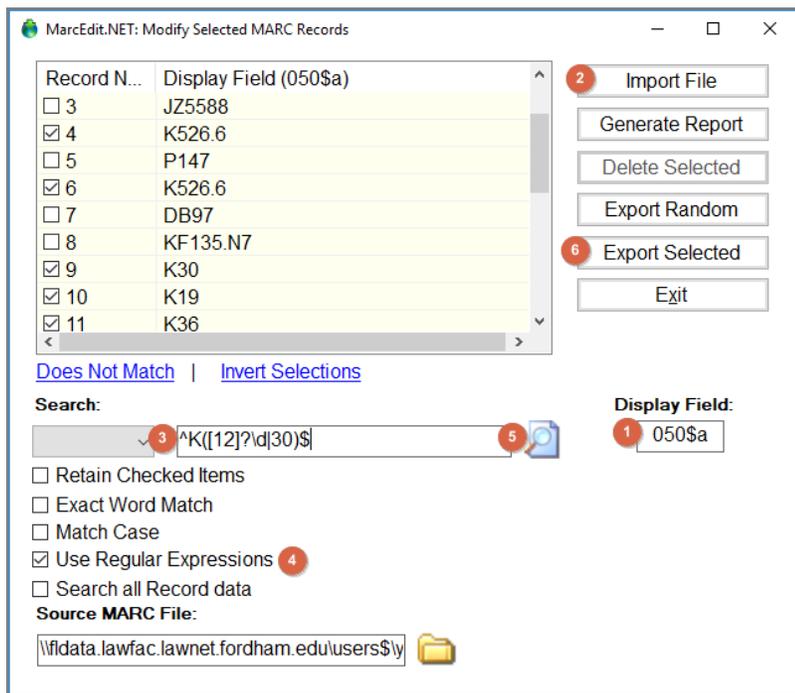
## SCENARIO 5: EXTRACTING A SUBSET OF RECORDS BASED ON CALL NUMBER RANGES

**Description:** We would like to extract from a file of MARC records only those records for law reviews with call numbers between K1-K30.

**Solution:** In the MarcEditor, follow these steps:

### File > Select Records for Edit

- 1 In **Display Field** enter 050\$a
- 2 Click **Import File**
- 3 Enter your expression
- 4 Check **Use regular expressions**
- 5 Click the **magnifying icon**
- 6 Click **Export Selected** and rename your new file



We are already displaying 050\$a, so the expression can start directly with the call number range.

- ^ Beginning of subfield
- K Literal letter "K"
- ( Open parentheses for OR Subexpression
- [12]? Zero or one instances of "1" OR "2"
- \\d Any digit
- | OR
- 30 Literal number "30"
- ) Close parentheses
- \$ End of subfield "a"

Putting it together: `^K([12]?\\d|30)$`

The caret and dollar sign set important boundaries. Without them we could end up with numbers that have K in a place other than the beginning of the subfield, and call numbers with more than two digits.

## THE PROCESS OF DEVELOPING THE ABOVE EXPRESSION

`^` = beginning of line

`K\d` = K with any single digit (K1, K2, K3, etc.)

OR

`K[12]\d` = K with any two digits starting with "1" or "2" (K10, K11, K12, etc., or K20, K21, K22, etc.)

OR

`K30`

`$` = end of line

Putting it together:

`^(K\d|K[12]\d|K30)$`

This expression works, but is longer than it needs to be. To simplify it, let's take the "K", which appears in every part of the expression, and use it only once outside the parentheses:

`^K(\d|[12]\d|30)$`

To make it even more elegant, we can condense the expression for single and double digits using the question mark, which specifies that [12] will appear zero or one time. In other words, for single digit K numbers, the [12] will be ignored and only the \d (any digit) will be captured:

`^K([12]?\d|30)$`

### Exercise 14

**Description:** We have a big file of records for vendor ebooks, but we want to load into our system only those titles with K and J call numbers. How can we extract these records?

Using **File > Select Records for Edit**, write an expression that will identify all call numbers starting with "K" or "J" and will ignore all other call numbers.

Try this exercise two ways:

1. In Display Field, enter **050\$a**
2. In Display field, enter **050**

How does each option affect your expression?

## Regular Expressions (regex) Commonly Used in MarcEdit

**Metacharacters:** . ^ \$ | \ ( ) \* + ? { } [ ] -- These characters need to be escaped with a backslash (\) to match literal characters

### Anchors & Escapes

<b>^</b> (caret)	Matches beginning-of-line position	<b>^cat</b> matches <b>cat</b> , <b>catalog</b> , but <b>NOT scatter</b>
<b>\$</b> (dollar)	Matches end-of-line position	<b>ght\$</b> matches <b>night</b> , <b>bright</b> , but <b>NOT lightning</b>
<b>\</b> (backslash)	Specifies the next character as either a special character or a literal	<b>\\$a</b> matches <b>\$a</b>
<b>\n</b>	Matches new line	<b>.\n.</b> matches This is first line; This is second line

### Character Classes

Only ^ - \ ] need to be escaped inside a character class

<b>[abc]</b> (Same as <b>(a b c)</b> )	Character set. Matches any one of the enclosed characters.	<b>s[mc]</b> matches <b>small</b> , <b>scroll</b>
<b>[^abc]</b>	Negative character set. Matches any character NOT enclosed	<b>c[^au]t</b> matches <b>cot</b> but <b>NOT cat</b> or <b>cut</b>
<b>[ - ]</b> (hyphen with square brackets)	A range of characters	<b>[a-zA-Z]</b> matches any lowercase/uppercase letter <b>[0-9]</b> matches any digit
<b>.</b> (period)	Matches any single character	<b>b.g</b> matches <b>big</b> , <b>bag</b> , <b>bug</b>
<b>\w</b> Same as <b>[a-zA-Z0-9]</b>	Matches any alphanumeric	<b>1\w3</b> <b>123</b> <b>1a3</b> <b>1B3</b>
<b>\W</b> Same as <b>[^a-zA-Z0-9]</b>	Matches any non-alphanumeric	<b>1\W3</b> <b>1%3</b> , <b>1+3</b> , <b>1\$3</b>
<b>\s</b> Same as <b>[ ]</b>	Matches whitespace	<b>\.\s{2,3}[a-z]</b> matches 1. <b>xyz</b> 2. <b>xyz</b>
<b>\S</b> Same as <b>[^ ]</b>	Matches non-whitespace	<b>.*\.\S.*</b> matches space missing between period and beginning of new sentence, e.g., <b>abc.Xyz</b>
<b>\d</b> Same as <b>[0-9]</b>	Matches any digit	<b>1\d3</b> matches <b>103</b> , <b>113</b> , <b>123</b> , <b>133</b> , etc.
<b>\D</b> Same as <b>[^0-9]</b>	Matches any non-digit	<b>1\D3</b> <b>1a3</b> , <b>1@3</b> , <b>1B3</b>

### Quantifiers & Alternations

<b>*</b> (asterisk)	Matches preceding expression zero or more times ( <b>.*</b> matches any character zero or more times)	<b>il*e</b> matches <b>mile</b> , <b>pier</b> , <b>grille</b> <b>12*5</b> matches <b>15</b> , <b>125</b> , <b>12225</b> , etc.
<b>+</b> (plus)	Matches preceding expression one or more times	<b>ar+</b> matches <b>are</b> , <b>arrow</b> , <b>carrier</b> but <b>NOT apple</b> <b>12+5</b> matches <b>123</b> , <b>1225</b>
<b>?</b> (question mark)	Matches the preceding expression zero or one time	<b>mo?r</b> matches <b>more</b> , <b>ramrod</b> but <b>NOT mooring</b> <b>12?5</b> matches <b>15</b> , <b>125</b>
<b>{n}</b>	Matches preceding expression exactly n times	<b>12{3}5</b> matches <b>12225</b>
<b>{n,}</b>	Matches preceding expression n or more times	<b>12{3,}5</b> matches <b>12225</b> , <b>122225</b> , <b>1222225</b> , <b>12222225</b> , etc.
<b>{n,m}</b>	Matches preceding expression between n and m times	<b>12{3,5}5</b> matches <b>12225</b> , <b>122225</b> , <b>1222225</b>
<b> </b> (pipe)	Or	<b>cat dog</b> matches <b>cat</b> , <b>dog</b>

### Grouping (in Find) & Substitution (in Replace)

<b>( )</b> (parentheses)	Group in Find. When matching a pattern within parentheses, we later use \$1, \$2, etc. in Replace to refer to the previously matched pattern	Reverse Last Name, First Name  Find: <b>([A-Z][a-z]+)</b> , <b>([A-Z][a-z]+)</b> Replace: <b>\$2[space]\$1</b>
--------------------------	--	---

